CHARACTERISTICS OF WEB-BASED TEXTUAL COMMUNICATIONS

A DISSERTATION SUBMITTED TO

THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Tayfun Küçükyılmaz

December, 2012

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Cevdet Aykanat(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Dr. Berkant Barla Cambazoğlu(Co-Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Fazlı Can

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Enis Çetin

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Ismail Hakkı Toroslu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Özgur Ulusoy

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

CHARACTERISTICS OF WEB-BASED TEXTUAL COMMUNICATIONS

Tayfun Küçükyılmaz
PhD in Computer Engineering
Supervisor: Prof. Dr. Cevdet Aykanat
December, 2012

In this thesis, we analyze different aspects of Web-based textual communications and argue that all such communications share some common properties. In order to provide practical evidence for the validity of this argument, we focus on two common properties by examining these properties on various types of Web-based textual communications data. These properties are: All Web-based communications contain features attributable to their author and reciever; and all Web-based communications exhibit similar heavy tailed distributional properties.

In order to provide practical proof for the validity of our claims, we provide three practical, real life research problems and exploit the proposed common properties of Web-based textual communications to find practical solutions to these problems. In this work, we first provide a feature-based result caching framework for real life search engines. To this end, we mined attributes from user queries in order to classify queries and estimate a quality metric for giving admission and eviction decisions for the query result cache. Second, we analyzed messages of an online chat server in order to predict user and mesage attributes. Our results show that several user- and message-based attributes can be predicted with significant occuracy using both chat message- and writing-style based features of the chat users. Third, we provide a parallel framework for in-memory construction of term partitioned inverted indexes. In this work, in order to minimize the total communication time between processors, we provide a bucketing scheme that is based on term-based distributional properties of Web page contents.

Keywords: Web search engine, result caching, cache, chat mining, data mining, index inversion, inverted index, posting list.

ÖZET

WEB TABANLI YAZILI İLETİŞİM KARAKTERİSTİKLERİ

Tayfun Küçükyılmaz
Bilgisayar Mühendisliği, Doktora
Tez Yöneticisi: Prof. Dr. Cevdet Aykanat
Aralik, 2012

Bu tezde, Web tabanlı iletişim metotlarının farklı özelliklerini inceleyip, değişik iletişim metotlarının ortak karakteristikleri olduğunu öne sürdük. Bu tezimizi kanıtlayabilmek için bu ortak özelliklerden iki tanesinin üzerinde yoğunlaşacak ve bu özellikleri derinlemesine inceleyeceğiz. Bu özellikler: Bütün Web tabanlı iletişim metotları yazarlarına,alıcılarına, veya mesajların kendilerine atfedilebilecek özellikler taşırlar. Ve bütün Web tabanlı iletişim metotları benzer dağılımsal özellikler gösterirler.

Bu iki hipotezi kanıtlayabilmek amacıyla üç farklı, pratik, gerçek yaşamla ilgili araştırma problemi üzerinde durduk ve bu iki hipotezi kullanarak sunulan araştırma problemlerini çözmeye çalıştık. Bu problemlerden ilkinde, halihazırda kullanılmakta olan bir sorgu motoru için sorgu özelliklerine dayanan bir otomatik öğrenme yaklaşımı öne sürdük. Bu çalışmada, kullanıcı sorgularından çeşitli özellikler çıkartarak bu özellikleri otomatik öğrenilmiş bir model oluşturmak için kullandık. Bu modele göre her sorguya bir kalite metriği atayarak, arama motoru ön belleğine kabul ve atılma kararlarını bu metrik sayesinde yaptık. İkinci problemde, kullanıcı ve mesaj özelliklerini tahmin etmek amacı ile bir chat sunucusunun verilerini inceledik. Sonuçlarımız birçok kullanıcı ve mesaj bazlı ozelliğin tahmin edilebilirliğine ışık tuttu. Üçüncü çalışmamızda, terim bazlı ters indekslerin hafıza bazlı ve paralel olarak oluşturulmalarını inceledik. Bu araştırmada ise, işlemciler arası toplam iletişim zamanını minimize edebilmek amacı ile, Web sayfalarındaki terimlerin dağılımsal özelliklerini temel alan bir guruplama metodu önerdik. Bu özellikleri kullanarak, işlemciler arası iletişim zamanını, işlemci görev dağılımını da dikkate alacak şekilde nasıl azaltabileceğimiz yönünde araştırmalar yaptık.

Anahtar sözcükler: Arama Motoru, Sonuç ön belleği, ön bellek, Chat madenciliği, veri madenciliği, indeks tersleme, ters dizin.

Acknowledgement

I would like to thank Berkant Barla Cambazoglu for his constant help, support and mentorship throughout this study.

I would also like to thank Prof. Dr. Fazli Can and Prof. Dr. Varol Akman for believing me and supporting me at every step of this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A Cohesion of Definitions: Communication and Knowledge Dissemination

It is widely believed that the first use of written language originates to Mesopotamia around 3200 BC. At that time, the use of writing was either to keep track of valuable resources such as grain or beer, or to preserve memorable events. For a very long time, writing is used solely to preserve the available information and pass it to next generations. Around 500 BC, writing has started to be used for a completely different reason: communication. First written communique according to the testimony of ancient historian Hellanicus the first, is a hand written letter by Persian Queen Atossa daughter of Syrus, mother of Xerxes. Although not an invention by itself, the use of writing

as a means of communication is a ground breaking event for the human kind that still affects our lifestyle.

From the first appearance of letters in human history up until mid 1940's, writing is used only for two distinct purposes: as a means of communication (such as using letters or telegrams), or for sharing and protecting knowledge (e.g., books, glyphs and etc.). When we examine post-40's writing style, communication-oriented writings have several distinctions from other literary products. First and foremost, all communication-oriented writings target a person or a position, which implies a degree of intimacy (acquaintance) between two peers. An even more important distinction due to this intimacy is that, these writings are generally accepted as a private communication media and involve some sort of secrecy between communicating peers. Even today, social custom dictates that we seal envelopes when posting letters as a courtesy of privacy.

With the development of computers, the mankind finds new means to store valuable information. Instead of writing on paper, papyrus, or inscribing on temple walls, computers allow information to be kept as electrical states, without physical limitations or constraints. Just as in the case of the invention of the written text, the use of computers as a means for communication followed the introduction of computers as a knowledge storage medium. HERMES, the email system built within ARPANET, was one of the first attempts of mankind for using computers as a means to communicate. Still, the use of computers both as a storage and a communication medium was

not very different from the paper-based methods, up until mid-1980's, until when the Internet emerges.

The Internet phenomenon arose in mid 1990's, when small networks start to merge with each other, and the World Wide Web start to be available to common people. Throughout the world, millions of people start to connect to Internet, building the worlds' largest society. Starting as a huge interactive knowledge repository, Internet also rapidly assumed the mantle of a communications medium. However, it was evident from the first day that, letters or mails, as the sole method of traditional textual communications, would both be unsuitable and insufficient means of communication for such a large community. Thus, the community has devised its new ways to communicate.

As the Internet community grows larger and larger, people become acquainted with a lot of new terms such as "forums, bulletin boards, and blogs." While these terms are derived from the physical world, the Internet community assigns them whole new meanings. With such communication platforms, people easily access expert knowledge and opinions, share their personal feelings and thoughts, and even create new relationships.

The most significant aspect of this new form of communication is twofold. First, the intimacy and privacy aspects of the traditional textual communications become extinct in this new communication media. Most of the dialogs on such platforms

cannot be classified as peer-to-peer communications, but rather peer-to-community communications. Trust and privacy in these media are rather targeted towards a self-constructed community instead of individuals. Second, such indirect communication methods combine both objectives of traditional writing: both writing as a knowledge media, and writing as a communication media. In fact, what is happening at the current time is that we, the new Internet generation, are assigning a completely new meaning to communication. Today, communication through the World Wide Web does not only mean a conversation between two peers, but also a textual life experience within a community. It also encapsulates the knowledge sharing phenomenon of traditional writing, making everyday conversation a more elite and complex issue.

In this thesis, we address this amalgamation of communication and knowledge dissemination. Throughout this text we will call this communication and information sharing phenomenon the “Internet communication” and try to prove that every Internet communication method shares some common properties. In order to prove our claim, we will provide three works from various areas of computer science, each of which is performed on different types of communication platforms.

Before presenting these varying works, we will first provide a background on the differing communication platforms over a taxonomy of today’s communications. Then we continue by establishing the common properties of different communication platforms, and present the connections of the works presented in this thesis with the acclaimed properties.

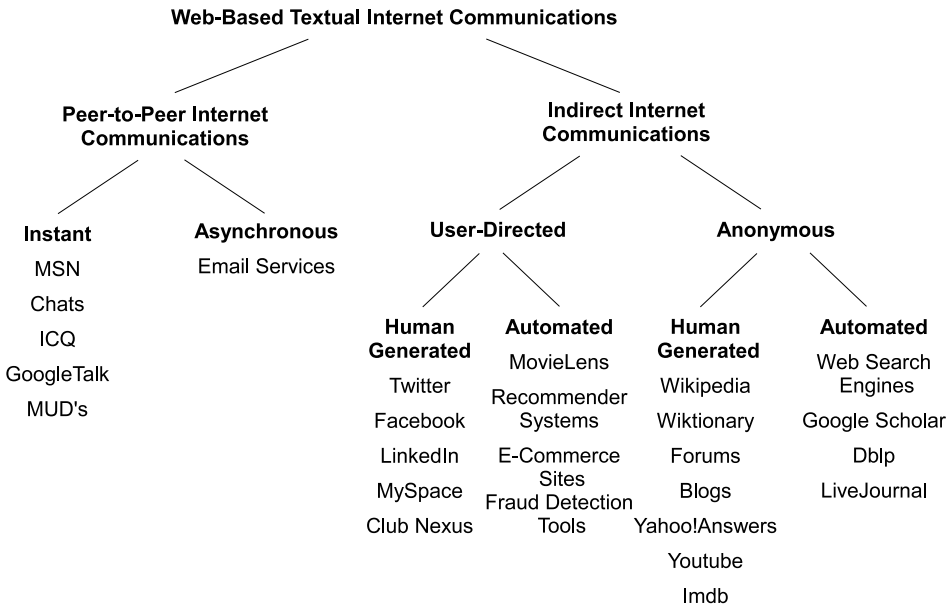


Figure 1.1: A taxonomy of Web-based textual communication media.

1.2 Background

The number of proposals and presentations about Web-based textual Internet communications in the literature is so vast that it would be a futile attempt to list even the mainstream publications. Instead we try to provide a classification of the textual Internet communications on the Web. Figure 1.1 provides a taxonomy of Web-based textual communications media. In this taxonomy, we first categorize textual communications media into two according to its target audience: peer-to-peer and indirect.

In peer-to-peer communications, each message/dialog is instantiated by a specific user, and each user message/dialog is written to target a distinct recipient. Commonly, the aim of peer-to-peer communications is to contact and converse with an acquaintance. This conversation can be on a real time basis similar to a face-to-face talk, or

without any timely obligation like writing a letter. Thus, peer-to-peer communication media can be further classified according to their temporal features as instant peer-to-peer and asynchronous peer-to-peer communication media.

In instant peer-to-peer communications, users can involve in real time textual conversations over the Internet. The Microsoft Network (commonly known as MSN) (89), Google Talk (Gtalk), Instant Messaging Computer Program (icq) (155), chatting servers (8; 81; 82; 118), and multi user dungeons (MUD's) are very well known examples of such communication platforms.

In asynchronous peer-to-peer communications, the intent of the user is to transmit a message to another user. The most well known type of these communications media is emails (19; 22; 39; 40; 78; 130; 141; 144; 149) A large number of work has been conducted on email messages and mailing platforms. Some of the streamline topics of these research are focused on writing style analysis (149), author characterization (141), author attribution (39; 144), social network mining (19; 22; 78; 130), and forensic studies (40).

In indirect textual communication platforms, users communicate through accessing/producing published data. In this sense, indirect Internet communications resemble a knowledge sharing activity more than a communication activity. In indirect communication platforms, the accessible information could either be created by a user,

or published automatically according to the needs of the users by means of a computerized system. Thus, indirect communications, by their very nature, are all asynchronous. In these platforms, the use of the communication media is to disseminate information within a computer-mediated society, or as a contribution to a knowledge base over the Internet. According to the audience of these communications we categorize indirect communications into two: peer-to-community directed and anonymous communications media. Within these categories, indirect communications can further be categorized with respect to the author of the data: human generated or automated communication platforms.

The peer-to-community directed human-generated communication platforms are generally social networking sites where people can rate their intimacy by declaring each other as friends, foes, or acquaintances. A common characteristic of such platforms is that they allow their users to create communities, and disseminate information within these communities. The asset of ability to create social communities is twofold. First it allows users to establish a trust with other information publishers based on common interests, and thus allow users to only browse trusted information. Second it facilitates two types of communication alternatives; either to communicate with the trusted users in a private manner, or disseminate knowledge publicly. Twitter (66; 67; 115; 137), Facebook (43; 93; 150), Myspace (43; 139; 140), LinkedIn (90), Club Nexus (3), and Slashdot Zoo (52; 85) are excellent examples of such platforms.

In user-directed automated communication media, through different statistical and

machine learning methods, the user patterns are analyzed in order to extract user-preferential information. As an illustrative example, the MovieLens (132) platform is a movie recommendation site, where the user ratings are gathered and analyzed in order to find user movie preferences. In the light of the findings, the site make recommendations to each user about upcoming movies. Recommender systems (84) and e-commerce (102; 112) sites are other examples of such communication platforms.

Anonymous human-generated communication platforms correspond to knowledge bases over the Internet. These knowledge bases are usually created and maintained by Internet users either as a community effort or individually. Various forums (2), blogs (1; 17), and bulletin boards fall into this category. With the recent popularity of such platforms, some professional, corporate funded versions of this media has also emerged. Some examples of such efforts are Wikipedia (123; 161), Wiktionary (161), Yahoo!Answers (90), Youtube (20; 51), and Internet Movie Database (IMDB) (68).

Unlike other platforms, the objective of anonymous automated systems is analyzing the already existing information base and facilitate user access instead of generating new information. The most commonly used example of anonymous automated systems is Web search engines (53; 157). Web search engines provide a means to ”dig out” existing information without consulting to an expert or doing exhaustive searches over the Internet. In order to provide sound responses to user queries, Web search engines catalog the whole Internet knowledge base, rank this knowledge base

according to each user query, and by using several analysis tools extract the most relevant results that would possibly satisfy the user requests. Other than Web search engines, several more specialized cataloging services also fall into this category according to our taxonomy. Co-authorship sites such as DBLP (3; 16), LiveJournal (16), and GoogleScholar (37) are examples of such specialized services.

1.3 Motivation

As Section 1.2 suggests, the versatility of Internet communication media is unparallel. However, literature also suggests that all communication platforms, and the text within such communications have common properties. These common properties vary from community graph-based properties (i.e. the connectivity of the community, the degree and radius of the communication graph), vocabulary-based properties (i.e. the variations of peer vocabularies and vocabulary distributions), to structural properties (i.e. heavy use of misspelling and noise due to anonymity of the communities).

In this work, we will concentrate on two of the most heavily used and exploitted properties of the Internet communication media and try to provide practical evidence that these properties hold no matter how versatile the communication structures are. These properties are:

- **Claim 1:**All textual communications contain characteristic markers inherent to its author and receiver.
- **Claim 2:**All textual Internet communications exhibit similar distributional properties. Here, as distributional properties, we refer to heavy tail distibutions exhibitted by both message logs and vocabularies of textual Internet communications.

In order to prove that these properties hold for all communication types, we provide three practical works on differing areas of computer science using data from differing communication media:

- AS the first problem, we examined whether it is possible to improve the performance of a query result cache for a search engine. To this end, we use the real life query logs retrieved from a commercial search engine.
- In the second work, we analyzed an efficient framework for constructing inverted indexes in a distributed environment. We use a dataset composed of

crawls seeded from the university sites in the USA in this work. The dataset contains more than 7 millin Web Pages.

- In the third work, we analyzed the chat message and user attributes on a chat message log. We use a peer-to-peer chat data retrieved from the logs of a university chat server. The data contains messages of over 1600 people during a 30 day period.

In this thesis, we have used three different datasets to verify our claims. In order to cover the presented taxonomy as wide as possible, we choose two datasets from Indirect anonymous automated Internet communication platforms and one dataset from peer-to-peer instant communication platforms.

As the indirect anonymous automated communications data, we used the query logs of a commercial search engine and a crawl dataset. The query log dataset is not publicly available and we are not permitted to disclose data specifications in this thesis. The crawl dataset is created by downloading the contents of html pages starting from several university sites in the United States. The raw size of this dataset is 30 GB. As the peer-to-peer instant communications data, we used the conversation logs of 1616 people using a local chat server which originates in Bilkent university, Turkey. The dataset contains more than 200,000 chat messages between various peers.

The rest of this thesis is organized as follows:

In chapter 2, we will provide a discussion about a machine learning approach for

the query result caching using features extracted from the user queries submitted to a commercial search engine. The problem of caching on a Web search engine can be summarized as follows: A critical observation about Web search engines is that, the query load of a Web search engine follows a heavy tail distribution. That is, a small subset of queries are frequently submitted to the search engine, while most of the distinct queries are submitted only once or no more than a couple of times. Given a set of previously submitted queries and their results, by storing the frequent queries in memory, it is possible to respond to a majority of future queries by using just memory references.

The very nature of the caching problem requires that there should be some queries that are more frequent than others. In fact, these frequent queries should be "common enough" to compensate the computational costs. Thus, the work presented in Chapter 2 would provide an insight on claim 2. Additionally, in this work, we provide a methodology to improve the hit rate of the cache by using features extracted from user queries. In this sense, the presented work exploits the user query-based characteristics in order to improve cache performance and thus verifies claim 1. In this chapter our contributions are as follows:

- First, we apply a machine learning approach to the query result caching problem. To this end, we attempt to predict the next arrival time of each query and use this perdiction as a quality metric.

- Second, in our machine learning approach, we used and evaluated an extensive set of features and examine the importance values of different features in the caching problem.

- Third, we identified several different class labels for our machine learning model and evaluated their predictability and usefulness for the caching problem.

- Fourth, we conducted our experiments on a realistic search engine data. We also discussed the results of the previous works, and evaluated their applicability on realistic datasets.

- Fifth, we applied our approach to both static caching and dynamic caching and evaluated its effectiveness.

- Sixth, during the analysis of static and dynamic caching, we present several accurate optimality conditions for both caching methods, and identified the possible room for improvement.

- Last, we applied our findings on a state-of-the-art caching framework with both static and dynamic components and present our results.

In Chapter 3, we provide a chat mining framework, where we question whether several user and message attributes are predictable by using text based features of instant messaging conversations. Some of the examined user and message attributes are: the author of a message, the receiver of the message, the horoscope of the user,

the educational level of a user and etc. We have used several term-based and writing style-based features in order to prove the predictability of user and message attributes. The results of this work would be used to verify claim 1. Our contributions in this work are as follows:

- To the best of our knowledge, the presented work is the first attempt to analyze online chat messages in the literature.
- We propose a chat mining framework to analyze online chat messages. Our framework also includes methods for analyzing very short chat messages and dealing with several data imbalance problems.
- We analyze both user-specific and message-specific attributes of chat messages and their predictability.
- We used both term-based and writing style-based features to summarize and examine the predictability of chat user and message-specific attributes.

In Chapter 4, we present a memory-based parallel inverted index framework. In a nutshell, index inversion problem can also be formulated as a matrix transpose problem, where the transposed matrix would be a term-document matrix. In a parallel formulation of the index inversion problem, the naive approach would be to transpose local term-document matrices, find a suitable storage setting and communicate the local indexes among processors. In this work, upon careful examination of this

model, we realized that the communication of all local vocabularies to a server machine would create a bottleneck and slow the communication considerably to a point that naive approach would be inapplicable for real life systems. Thus, we exploit distributional properties of the term-document matrix and propose a bucketing strategy. In this sense, the success of the proposed scheme hints to the correctness of claim 2. In this chapter our contributions are:

- We propose an in-memory parallel inverted index construction scheme and compare the effects of different communication-memory organization schemes to the parallel inversion time.
- We propose a method to avoid the communication costs associated with global vocabulary construction which also eliminates the need of creating a global vocabulary completely.
- We investigate several assignment heuristics for improving the final storage balance, the final query processing loads, and the communication costs of inverted index construction.
- We investigate the effects of various communication-memory organization schemes.
- We test the performance of the proposed schemes by performing both simulations and actual parallel inversion of a realistic Web dataset and report our observations.

In Chapter 5, we first discuss each presented work in this thesis seperately, summarize their scientific contributions and comment on possible future directions of these works. Next, we repeat our remarks on how we exploit our claims within these works and conclude by explaining about how these works can be perceived as proof for our claims in these thesis.

Chapter 2

A Machine Learning Approach for Result Caching

2.1 Introduction

Today, Web search is the most dominating method for finding and accessing knowledge. As the volume of information on the Web grows larger, it becomes almost impossible to find relevant Web documents manually. Web search engines alleviate this problem by providing their users an easy way to access any information over the Internet. However, considering the sheer volume of data on the Internet and the growing number of Web users, responding to all user requests within a reasonable time interval is not an easy task. In order to respond to user queries, a search engine must identify

the relevant pages, rank them in relevance order and present the resulting set of pages. All these operations should be carried out in a short amount of time before the Web user loses interest to the result of his/her query. On the other hand, from the standpoint of a Web user, the advances in networking and computational technologies are generating an even increasing demand for faster and more precise query results from the Web search engines.

In order to meet these high access latency and throughput requirements of the Web community, Web search engines employ several performance improvement techniques. One of the most commonly used techniques for improving the search engine performance is caching. Caching is motivated by the repetition tendency of popular queries and the resulting high temporal locality of the user queries. The idea of caching is straightforward. By storing only a small portion of the most commonly accessed data in memory, a search engine can respond to future references of user queries without wasting too much computational and networking resources.

Aside from its immediate benefits, caching could also be an asset for a search engine in multiple perspectives: First, by reducing the data to be transmitted to the servers, it reduces the network load of the search engine. Second, it reduces user perceived delays by eliminating computation time that need to be spent on a query. Third, by reducing the computational load on the server side, it enables higher throughput. Last, it provides higher availability since cached data can also be used as a replica of the original data regardless of availability constraints (32).

Temporal locality in the caching problem for Web search engines manifests itself in two forms: as recency and frequency. Recency describes the bursty behavior of user queries. In that respect, a query that is submitted to a search engine is likely to be submitted again within a very short time interval. As an illustrative example, this behavior can be best explained by query submissions before the premiere of a new sensational movie. It would be reasonable to expect that many people would search about the movie or its' cast right before it is shown in theaters. But after a couple of weeks, the number of related queries start to decrease since most people have already watched it. Frequency , describes the steady behavior of user queries. Some queries, because of their general popularity, tend to be submitted more frequently than others. For example, navigational queries directed to social network sites, shopping sites, and Web search engines tend to cover a large proportion of the overall query load of a Web search engine. Thus, it is reasonable to expect that such queries will be submitted repetitively over long periods of time.

Although recency and frequency of user queries are major underlying features for the caching problem, none of these two features have superiority over the other in terms of caching. Past works (45) on caching show that, a combination of both works best as a state-of-the-art caching strategy in Web caching. It is also stated in literature (13; 50; 87) that Web search queries can be mined to extract several features that are not directly related to temporal locality, but can still improve the effectiveness of caching. As an illustrative example, it is reasonable to assume that short queries have a higher

probability of reoccurring than longer queries. For the sake of this example, a good cache replacement policy should also take query length into account. In that sense, for an effective caching strategy, not only recency and frequency, but other features should be incorporated into one policy.

In this chapter, we propose a machine learning approach to find a "good" caching policy which incorporates several different aspects of query result caching. Our main objective is to find a method for incorporating both recency and frequency, as well as several other valuable features, into a caching policy. To this end, we first define each query as a set of representative features extracted from the user queries. In our approach, instead of a recency- or frequency-sorted cache, we use a machine learning cache, where we try to predict the next re-occurrence (next arrival time or IAT-Next) of each user query and use this information as the cache replacement policy. In that respect, the work presented in this chapter is the first attempt in literature to use a machine learning approach as a cache replacement policy. Recency and frequency, as two major caching policies, are also incorporated into our approach as a set of representative features.

The organization of this chapter is as follows: In Section 2.2, the previous work on caching is presented, focusing mainly on the query result caching. In section 2.3, the machine learning approach in this work is presented. Specifically, the features and class labels that are used in this work are presented. In section 2.4, the dataset and experimental setup are explained.

In Section 2.5 and 2.6, we look at the two extreme cases for result caching: First, in Section 2.5 we analyze the effectiveness of result caching when the cache is fully static. Then in Section 2.6, we evaluate the other extreme, when the cache is fully dynamic. In these two Sections, different static and dynamic result caching methods and optimality conditions of both approaches are analyzed, and application of the proposed machine learning approach to both cases are examined along with experimental results. In Section 2.7, we combine static and dynamic caching approaches into one. We take the state-of-the-art static-dynamic cache (SDC) (45) as a baseline method, and apply our machine learning strategy on SDC. We present an extended discussion on the result caching problem and the results of our experiments in Section 2.8.

2.2 Related Work

For search engines caching can be employed on different data items such as the posting lists, precomputed scores, query results, and documents (111). The literature mainly concentrate on two of these data items: storing the posting lists and storing the query results. Apart from these works, several hybrid models are also proposed in literature. In (111), the authors propose a five-level caching architecture for different data items and propose methods to adress dependencies between the cached data items. In (12), the problem of storing posting lists, query results, and the list intersections are examined on a static cache setting. The work of (95) and (129) concentrate on the similar

problem on a dynamic setting. In another work (99), the similar problem is examined on a parallel architecture. In (134), the authors concentrate on pruning posting lists and storing these pruned lists to conserve cache space.

The posting list caching (12; 14; 143; 162) corresponds to storing the inverted lists of query terms in memory. The aim of posting list caching is to avoid disk accesses and computations required to calculate the relevant query results. Since posting list sizes follow a Zipfian (164) distribution for the Web data, it is possible to answer a large number of queries by just storing a limited number of posting lists (12). However, even when all the posting lists required to answer a query are stored in memory, these posting lists may need to be combined to achieve final results, which would still require additional computational power. Thus, even though high hit rates are easily possible for posting list caching, the computational gain would be limited.

Query result caching (5; 6; 13; 45; 87; 88; 98; 101; 110; 133) corresponds to storing the answers of a particular query in memory. The aim of query result caching is to exploit temporal locality of popular queries and respond to later queries by using pre-computed answers. Since a query result cache hit requires an exact matching of the incoming query and the query that is in the cache, the hit rate of a query result cache is lower than that of a posting list cache. However, when a hit occurs, the results can be directly answered by the result cache, and thus no additional computation is required. In this work, we focus on caching the query results. Thus, in this section, we will concentrate on the proposals about query result caching in literature.

The methods for improving the efficiency of query result caching in literature can be categorized into four classes according to policy decisions employed during caching: admission, eviction, prefetching, and refreshing. Admission (13) relates to giving a decision about whether to cache a query or not, based on a quality metric. The main purpose of admission is to identify queries that would pollute the cache and act as if those queries were never submitted. Eviction (6; 12; 15; 45; 50; 98) corresponds to selecting queries that are least likely to get a hit in the near future in order to provide space for admitting newer query submissions. As a baseline eviction policy, recency feature (evicting the least recently used query (LRU)) is widely adopted in literature.

Usually in most search engines, a query returns top 10 most relevant results to the user. The search engine-generated response page containing the links to these relevant pages is often referred to as a result page. For a search engine, sometimes it could be more beneficial to admit more than only one result page to the result cache. Prefetching (45; 87; 88; 100; 101) policies are used to decide how many of the result pages would be most beneficial to store in the result cache while admitting a query to the cache. This way, if a user requests the results of more than one page, the results would be returned without any extra cost. The main drawback of prefetching is that selecting an optimistic policy would pollute the cache with results that would never be required which would waste cache space.

Refreshing (24; 25; 31; 125; 126) aims to improve the hit rate of the result cache by improving the freshness of the already existing results stored in cache. The main

motivation behind refreshing is that the contents of the cache might get older in time and would not be able to serve as adequate answers to user queries. A refreshing policy decides which query results should be re-fetched from the Web so that the freshness of the cache contents are preserved and that, the Web search engine does not provide users outdated information.

In this work, our aim is to find a caching policy by employing machine learning methods to the query log, so that the hit rate of the query result cache is improved. To this end, we use a static-dynamic cache assumption. In this method, the cache is divided into two segments; a static segment and a dynamic segment. For the static segment, we use our machine learning approach to find a quality metric among user queries, and use that metric to fetch the most beneficial set of queries to fill the static cache. For the dynamic segment, we use our machine learning approach as an eviction policy, to find the least beneficial query within the dynamic cache and to evict that query in order to provide cache space for more recent, and possibly more beneficial submissions.

Our proposed method is motivated by several works in the literature. Throughout this work, we also adopted some of the past proposals, use them as baseline for comparison, and evaluated their performances on a real life setting. We also feel that, some of these works should be mentioned due to the parallelism in their approaches to the caching problem.

The work of (98) examines the query result caching for the first time in the literature. In (98), the author evaluates the effectiveness of result caching with four recency based eviction policies. The author also emphasizes the importance of frequency for caching, and for the first time in literature, proposes a static caching scheme. According to his proposed scheme, a static cache is composed queries with the highest frequency in the training set.

In (45), the authors, proposed the partitioning of the result cache into two segments: a static segment and a dynamic segment. In their proposed model (SDC), the static cache is filled with the most frequent queries using a query log while the dynamic cache uses a LRU-based eviction policy. In essence, the static cache responds popular (frequently submitted) queries while a small LRU-based dynamic component is used to respond to bursty query behavior. Today, most of the works in the literature accept SDC as a state-of-the-art caching policy and use it as a second baseline method along with LRU.

For result caching in search engines, in literature, there are several proposals that emphasize using feature-based approaches to exploit different characteristics of the user queries. In (13), the authors present a feature-based admission method for query result caching. In their proposed method, cache is divided into two parts: an admission cache and a controlled cache. Using the query length feature the proposed policy decides whether to admit the query into the admission cache or not. Remaining queries are admitted into the controlled cache, which is using the LRU policy.

In (110), the authors present another feature-based approach to improve hit rate of the static cache. In their approach, they define a stability metric, where stability is defined as the standard deviation of query frequency within discretized time intervals. In this method, a low standard deviation means that query is more likely to be received again than other queries with higher deviations. Instead of admitting most frequent queries into the static cache, they filled the cache with most "stable" queries.

Another feature-based result caching architecture is presented in (50). In their work, the authors present a fully dynamic feature-based result cache eviction scheme. Using several query-based features, the authors classify each incoming query into "query buckets", where each query bucket is essentially a LRU cache segment. Then they prioritize these buckets with respect to their relative hit rates and evict queries in from the bucket with the least hit rate. In that respect, the hit rate of a bucket can be considered as a quality metric for that bucket.

Machine learning methods are also used in result caches in the literature. In a recent article, (126) applied machine learning methods on query result caching. In their work, th authors propose a machine learned cache invalidation technique is proposed-for determining whether a query result/posting list is fresh or stale. In their approach, the authors train a machine learning model in order to predict time-to-leave (TTL) values for each query occurence. To this end, they use several query log features for training their machine learning model. In the sense that their proposed machine learning approach is applied to each query occurence in the query log, their approach is

similar to our proposed mechine learning method in this work.

As a feature-based result caching policy, our proposed machine learning cache has several differences from the previous works in the literature. First, we use several query-, frequency-, recency-, term-, and user-based features in order to learn a policy from the past query logs. In that respect, our proposed model not only incorporates query recency and frequency, but also exploits other characteristic markers of user queries. Second, our model enables a more flexible approach for caching, where the caching policy can be re-trained over time in order to reflect the changes in user and query bevavior. Third, it allows us to analyze the impact of different features on caching.

2.3 Machine Learning Approach for Result Caching

In this work, our aim is to find a "good quality metric" for user queries, which would encapsulate query recency, query frequency, and several other query characteristics. For this purpose, we model the query quality metric as the next arrival time (IAT-Next) of a query. In order to predict the IAT-Next of the queries, we model the result caching problem as a single-label regression problem, where next arrival time is the predicted class label. We experimented with several variants of IAT-Next using different machine learning tools and algorithms. In this section, we first describe the features used as variables in our machine learning approach, then we describe the class labels used in

our experiments.

2.3.1 Features

In this work, we used 30 different features extracted from a realistic query log. For a clear presentation, we classify these features into six categories. These categories are: query string-based, user-based, search engine related, term frequency-based, query frequency-based, and temporal features. Table 2.1 summarizes the features used in this work and their categories.

Query string-based features describe the structural properties of a query. We find query string-based features particularly important in this work, because in the literature, there were several works that use such features for improving cache performance (13; 50). These features are also quite popular in the caching research since these features are static. That is, they do not need re-processing since queries do not change feature values at every occurence. We use five such features. Query_Length is the query size in characters and Word_Count is the number of terms in a query. Is_URL_Present is a binary feature, which takes value 1 if the query string contains the sub-string “HTTP” or “FTP”, and 0 otherwise. Is_Domain_Present is another binary feature that gets the value 1 if the query contains any of the top-level domain names (94), and 0 otherwise. Average_Query_Term_Length is the query length divided by word count of a query.

User-based features describe general user behavior during the submission of the query. We use four user-based features. Is_User_Logged feature is the average number of users that are logged into their user accounts divided by the query frequency. Page_Number describes which page of the query result is requested/displayed by the user. Page_Number is a particularly interesting feature for our work since it encapsulates the essence of prefetching. Although beyond the scope of the work presented here, it is also possible to integrate prefetching in our proposed machine learning cache using the Page_Number feature. Click_Count is the average number of clicks users issue after getting the result of their query. We include this feature in our work, since it is closely related with the accuracy of the query responses of the search engine. First_Link_Click_Count is a subset of the Click_Count feature. It is defined as the average number of first link clicks issued per occurrence of a query. We expect that both accuracy and the popularity of a query could be exploited using both Click_Count and First_Link_Click_Count features.

Search engine related features describe attributes that are not directly visible to the user, but could still contain hints about the popularity of a query. Total_Number_Of_Hits is the average number of relevant result pages that is returned by the search engine to the user query. Note that, the number of hits is not a static feature since it is possible that the relevant pages may expand due to posting of new pages during testing or some servers may contain partial or outdated information. Rarest_Query_Term_Index_Size, Most_Common_Query_Term_Index_Size, and

Average_Query_Term_Index_Size are the minimum, maximum, and average posting list sizes of the query terms respectively.

The term frequency-based features describe the frequency related aspects of the query log. These features relate to the general popularity of queries and may infer the submission rate for each query over time. In order to incorporate frequency to our approach, as well as to detect possible variances in query frequency, we use a windowing mechanism. In our approach, we calculate the term frequencies of each query using its last one minute, one hour, and one day occurences in the query log.

Similar to recency, query frequency is another valuable feature for caching in the literature. In order to incorporate query frequency in our machine learning approach, we define four query frequency-based features. Like term frequency-based features, we use a similar windowing method. We define four time frames and calculate query frequencies within these time frames. These time frames are: query frequencies for the last minute, last hour, and the overall query log.

Temporal features describe the behavior related with submission time of a query. We define three different features for this purpose. Query_Submission_Hour is discretization of query time in hours in Greenwich timezone. Query_Day_Count is the average number of times a query is submitted during day time, where day time is defined as the interval between 7.00 AM to 19.00 PM. Query_Time_Compatibility is a binary classification for Query_Day_Count feature. After Query_Day_Count of a query

is calculated from the query log, queries are classified into three groups: day queries, night queries, and without timezone. If a query is submitted at night time more than 80% of the time in the observed portion of the query log, then it is considered as a night query. Similarly, if a query is submitted at day time more than 80% of the time in observed portion of the query log, it is considered as a day query. A query that is submitted at a time inconsistent to its timezone is counted as incompatible and given value 0, and in the latter case 1.

2.3.2 Class Labels

In this work, for predicting the next arrival time of the queries we used a two classifier-approach. First, we trained a singleton classifier in order to predict the singleton queries. Then, we train a second classifier with a training set where all singleton queries are removed and try to find a regression for the next arrival time (IAT-Next) of the remaining queries.

The rationale behind using a two-classifier approach is as follows: Since Web query logs follow a power law distribution (164), most of the distinct queries occur only once. However, singleton queries do not have inter-arrival times since they appear only once in the dataset, which makes such queries “uninformative” with regard to IAT-Next. Our experiments also showed that using singleton queries in the training set for predicting next arrival time of queries cause poorly predicted regression results.

In our approach, the first classifier maps each test instance to the interval (0,1) by fitting a regression model, where class label 0 in the training set means that the query is a singleton and class label 1 in the training set means otherwise. The results of the first classifier gives an estimate for each test query being a singleton or not. Throughout this work, we will refer to this classifier as "the singleton classifier".

The second classifier takes only the non-singleton queries in the query log for fitting a second regression model, where class label represents the estimated IAT-Next of a query. Throughout this work, we will refer to this second classifier as "the IAT-Next regressor". The class labels in the training set is constructed using the next arrival times of the queries within the training set. For the test set, the same label is the objective to be predicted by the machine learning methods. In our approach, this prediction would be used as the quality metric of a query during eviction; it would be most beneficial to evict queries that are expected to come later than others, since they are the expectation we infer from the predictions of the machine learning methods is that such queries will reside in the cache longest without producing any hits. Note that, in our approach, predicting queries as singleton or non-singleton is a subset of the latter problem in regard to the information to be predicted, since both classifiers compute a regression of the re-occurence time for each query. However, since the IAT-Next regressor make the predictions of IAT-Next for singleton queries in an almost-arbitrary fashion without any prior knowledge of singleton queries, in terms of instance size, singleton prediction is a superset of the latter problem.

In order to combine the results of these two classifiers we used two approaches. In the first approach, we used the singleton classifier as an admission policy. According to this admission policy, the queries that are predicted as singletons are eliminated directly before cache admission, while the result cache is ordered according to the next arrival time predictions. In the second approach, the predictions of the singleton classifier are used as support values for the regression model. That is, the results of both classifiers are multiplied in order to obtain the quality metric for each query. In this sense, the latter approach is an eviction policy for the query result cache. Our experimental results show that, using singleton query prediction as a support value perform consistently better than using it as an admission policy. For this reason, in the forthcoming discussions we will only refer to the second approach as our caching strategy.

For each of these classifiers, we experimented with four different class labels: The number of queries between two appearances, logarithm of the number of queries between two appearances, time in seconds between two appearances, and logarithm of time in seconds between two appearances of a query. Our experimental evaluations showed that all of these class labels perform almost equally in predicting the next arrival time of queries. However, experiments using the number of queries between two appearances as class label perform slightly better than other class labels. For the purpose of clarity, we will present the results of only this feature in the upcoming discussions.

2.4 Data and Setup

In order to examine the effectiveness of the proposed machine learning approach, we conducted extensive experiments on a realistic dataset constructed using query logs of a commercial search engine. Furthermore, as machine learning algorithms, we used several classifiers for training our result caching policy. In this section, we first introduce the query log and discuss the experimental setup that we have used during our experiments. Then, we present the classifiers that we have experimented with and discuss our criterion while selecting these classifiers.

2.4.1 Query Log and Experimental Setup

In order to verify our claims, we conduct our experiment on a query log constructed using submissions to a commercial search engine during 2011. For our experiments, we applied several preprocessing operations on the query strings. The punctuation marks in the queries are cleared, and query strings are normalized by converting all characters into lowercase. All query terms are rearranged in alphabetical order in order to eliminate dissimilarity as a result of term positions. Finally, spell correction is applied to the dataset.

In order to test our result caching approach and conduct our experiments, we divide the dataset into five phases. These phases are called training-warmup, training,

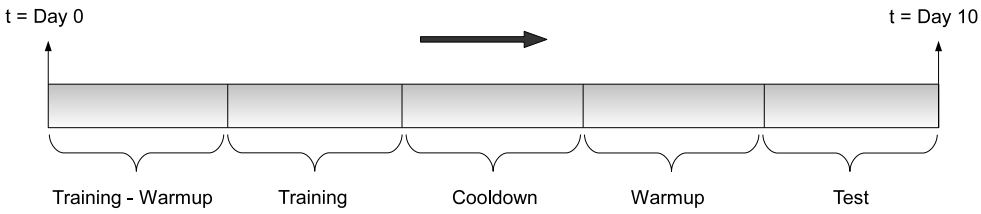


Figure 2.1: The division of the dataset in our experimental setting.

cooldown, warmup, and test phases. Figure 2.1 summarizes this separation on the query log. For a balanced partitioning of data among these phases while preserving the practicality of our approach for deployment on a real search engine, the dataset is divided as follows: The first day of the query log is used for warming up the query result cache for training, the next 6 days are used for training a machine learning model. The 8th day of the query log is used as the cooldown phase. The cooldown phase can be considered as the dual of the warmup phase. It provides the machine learning model a "future knowledge" so that the model can reflect to the fact that the data stream is infinite and the queries at the end of the training log may appear again in the future. The 9th day of the query log is used as warmup for testing and the last day of the query log is used as the test phase. In a practical deployment strategy, the aim of this division is to use an already existing query log for training a caching policy every day, and at the end of the day prepare a new caching policy for the forthcoming days in a pipelined fashion.

As noted earlier in Table 2.1, several features used in this work are defined over a

time frame such as term and query frequency-based features. Using the queries from the first day of the query log may lead to inconsistencies for such time-windowed features. The purpose of the training-warmup phase is to allow stabilization of these features. Queries within the training-warmup phase are skipped from the dataset for training purposes.

The queries in the training phase are used for generating the machine learned eviction policy. Each query in the training phase is labeled with its future interarrival time (IAT-Next), and we fit our regression model on these queries.

One important problem while generating a regression model in the training phase is that, the very last occurrence of every query in the training set would unavoidably marked with infinite next arrival timestamp due to the fact that the training set is finite and those queries would not be expected anymore. However, in a practical case, it is highly likely that many such queries would appear again in the future. The purpose of the cooldown phase is to provide the queries in the training phase a "finite future knowledge". This way, the last occurrences of queries in the training phase would be labeled reflecting their future behaviors.

The last two days of the query log is used as warmup phase and test phase respectively. For both phases, we use our fitted regression models in order to find a likelihood of future occurrences, and sort the result cache using these likelihood values. The queries in the warmup phase are first used for filling the cache in order to

prevent cold-start. Then queries in the test phase are used to evaluate the effectiveness of our algorithms.

2.4.2 Setup - Classifiers

We have used several machine learning tools such as Weka (57), Orange (41), Liblinear (46), and GBDT (159) to fit a regression model to our training data. We analyzed the results of several machine learning algorithms such as multilayer perceptron, pace regression, support vector machines, k-nearest neighbours algorithm, logistic regression, and gradient boosted decision trees in order to find the most suitable algorithm for our problem.

Our experiments showed that algorithms provided by both Weka and Orange are not suitable for evaluating large scale data due to their poor running time performances. In terms of efficiency, GBDT performed consistently better than Liblinear in all experiments. Thus, we choose gradient boosted decision trees algorithm provided by GBDT for training our proposed result cache eviction policy.

Gradient boosted decision trees (49) (GBDT) is one of the most widely used learning algorithms in machine learning today. Two appealing factors its popularity are that first, the results produced by GBDT are simple and interpretable; second, the models created by decision tree-based methods are non-parametric and non-linear. GBDT is a machine learning method based on on decision trees. It builds a regression model in

stages, by computing a sequence of simple decision trees where, each successive tree is built for refinement of the results of the preceding tree. In GBDT, decision trees are generally binary trees, where each tree is composed of decision nodes. The learning method is to find the most discriminative criteria for the data, use this criteria as a decision node, and recursively partition the data at each node of the tree.

For our experiments, we have used a version of GBDT based on the implementation of (159) which is currently deployed in some commercial search engines. After evaluating the effectiveness of GBDT with different number of decision trees and different number of nodes in each decision tree, both in terms of running time and regression accuracy, we set the maximum number of trees as 40 and the number of nodes in each tree as 20 in our experiments.

2.5 Static Caching

In this section, we analyze the effects of static caching for the result caching problem. First, we present several static caching methods already presented in the literature. Then, we propose two new methods for selecting which queries to admit into the static cache: A recency-frequency based approach and a machine learning based approach. Additionally, in order to better evaluate the room for improvement in static caching, we provide two new, and tighter, bounds for the optimality conditions of static caching.

2.5.1 Techniques

In the static cache assumption, the result cache should be filled prior to deployment of the cache. The basic strategy is to use a quality metric to evaluate/predict which queries would be more likely to come more frequent than others and construct the static cache using these queries. In this work, in order to evaluate the effectiveness of our proposed strategies we have implemented 5 different query selection strategies. In order to evaluate the room for improvement in static caching we also propose two new optimality conditions. These query selection strategies are:

Recency Sorted: Recency is the underlying metric for least recently used (LRU) caching strategy. The LRU heuristic assumes that queries not submitted recently will have a lower probability of getting submitted in the near future. LRU is considered as a baseline method in most previous caching literature.

Frequency Sorted: Frequency is the underlying metric for least frequently used (LFU) caching strategy. The LFU heuristic is closely related with temporal locality and is based on the assumption that the most frequent queries in the query log are also likely to exhibit a similar behavior in the future. That is, they are more likely to be submitted in the future.

Query Deviation Sorted: This strategy is based on the work presented in (110). In this work, the authors emphasize the fact that frequency-based strategies have the

disadvantage of under-valuing the bursty behavior in query traffic and propose a frequency stability metric. In this strategy, queries that are submitted in a more steady fashion are better candidates for the static cache.

In query deviation sorted caching strategy, in order to evaluate the stability of a query, first the query log is divided into constant length time frames. Then, the query submissions within each time frame is considered as a unit and submission variation between frames is calculated for each query. Queries having the least variation is considered as the best candidates for admission to the static cache. In our experiments, we selected a time frame of 1 day for evaluating the effectiveness of this strategy.

Recency + Frequency Sorted: During our evaluations we observe that, the strategy proposed in (110) suffers from the fact that it is possible to over-value queries that are observed infrequently but have very stable behavior. In this strategy, we adopted the proposed strategy and make several adjustments.

First, similar to (110), we divide the query log into unit time frames. We use a time frame size of 1 day for our experiments. Second, we normalize the frequencies of each query for each time frame using the total number of queries submitted during that time frame. Our motivation is that, since the total number of submissions within each time frame is not constant, a normalized query frequency would serve better for the stability of a query. Third, instead of using stability feature, we used query expected frequency as the admission metric. In our strategy, query expected frequency is equal

to the average normalized frequency of a query for one time frame. Finally, we make an attempt to combine the recency feature into our new strategy. To this end, we use an aging function during the calculation of query expected frequency. According to our strategy, the query frequency within a newer time frame has more effect than the query frequency within an old time frame. For example, a query that is more frequent recently but less frequent in the past is more valuable than a query that is less frequent recently but more frequent in the past.

Oracle - Test + Train: In order to propose a tighter optimal bound for the static caching problem we used the Belady's algorithm. According to Belady's algorithm, if, hypothetically, we have known all future occurrences of each query, we could have decided which queries to keep in the cache in the best possible way. That is, given a finite-sized cache, we could select the best set of queries to keep in the static cache. To this end, we calculated the query frequencies within the test set and pick the most frequent queries to use in the static cache.

Oracle - Train Only: Although the above strategy is optimal, it requires us to "clairvoyantly guess" the queries that have never occurred in the train set. In this strategy, we only picked the most frequent queries in the test set, only if they also occur in the training set.

Machine Learned: In this strategy, we used our proposed machine learning approach to the training set. We first fitted a regression model to all queries, where the

regression model predicts the next arrival time of the queries. According to this model, queries with smaller next arrival times are likely to come earlier than others during the test phase. Using these regression values, we calculated the estimated test phase-frequencies of each query and use this value as admission metric for the static cache.

2.5.2 Results

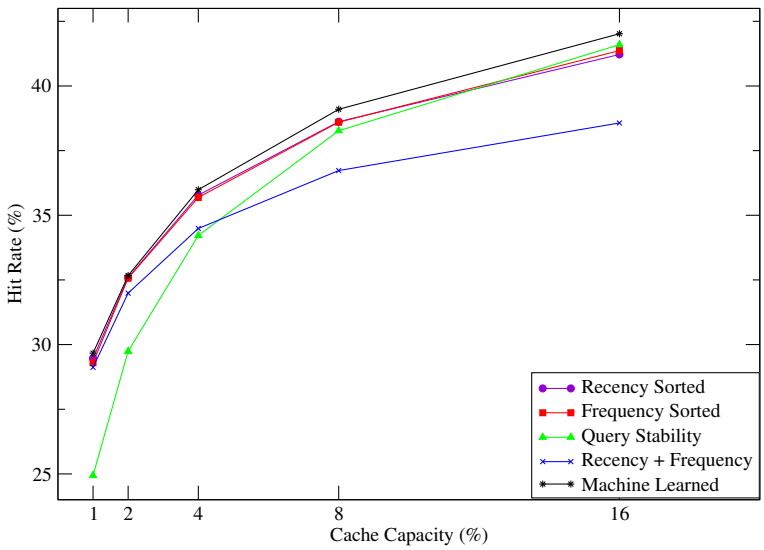


Figure 2.2: Performance of different static caching strategies for a fully static cache.

In order to evaluate the effectiveness of different static caching strategies, we perform several experiments. We use the hit rate of each algorithm as the performance metric. We have done experiments with each strategy on varying cache capacities. In our experiments, we selected the cache capacity as a function of the number of distinct

queries in the test set and used 1%, 2%, 4%, 8% and 16% of the test set as cache capacities. Figure 2.2 summarizes the performances of different static caching strategies on various cache capacities.

For the fully static caching problem, both LRU and LFU strategies perform almost equally well, where LRU perform better for the cache capacities up to 8% and LFU perform better with the 16% cache capacity. Reminding that recency feature is more suited for detecting bursts in query appearance, these results can be best explained by the fact that when the cache capacity is limited, increasing popularity of some queries overwhelm the frequency order. That is, keeping new queries instead of frequent queries have merits for a more effective caching. However, when the cache capacity is large enough, frequency feature start to acknowledge the popularity of such new queries and start performing similarly. And when the cache capacity is large enough recency feature start to degrade with respect to frequency due to cache pollution caused by singleton queries.

Among all policies, the query stability policy perform poorest for small cache sizes. This is due to the fact that several less frequent queries have significantly better stability values than some popular queries, polluting the static cache. However, when the cache size is large enough stability perform better than both LRU and LFU policies, since even with cache pollution there is enough cache space to accommodate the queries which are both stable and popular at the same time. The results of this experiment also show that, the real life search engine data and the observed query behavior in the

search engine data is quite different and much more robust than other datasets, and methods proposed over such data are not directly applicable to real life problems.

In general, our proposed recency + frequency policy perform very poorly. This is due to the fact that, even with normalization, the underlying feature of our method is a combination of query stability and query frequency and both methods perform poorly for small cache sizes. We also argue that combining recency and frequency into one policy is not a trivial task that require a more complex relation than query aging.

Among all methods, our proposed machine learning strategy perform consistently best for static caching. For small cache sizes the performance of the machine learned static cache is almost similar to other methods. However, as the cache capacity grows larger the improvement due to machine learned caching strategy become even more apparent. We can come up to two conclusions according to these results: First, machine learning is a viable way for combining both query recency and frequency into one strategy. Second, in addition to recency and frequency there may be other global characteristics of user queries that can be mined to facilitate caching.

Table 2.2 shows the 10 most discriminating features of the machine learning approach for static caching. Two of the top 3 most discriminating features is variants of query frequency which validates the importance of the frequency feature in caching. The Query_Time_Compatibility feature shows that our machine learning approach also identifies the fact that some queries are more susceptible to submission during certain

periods of time within a day. word count, page number, query length, and inverted index sizes of user queries are also identified as several other query characteristics that may be closely related to the popularity of a user query.

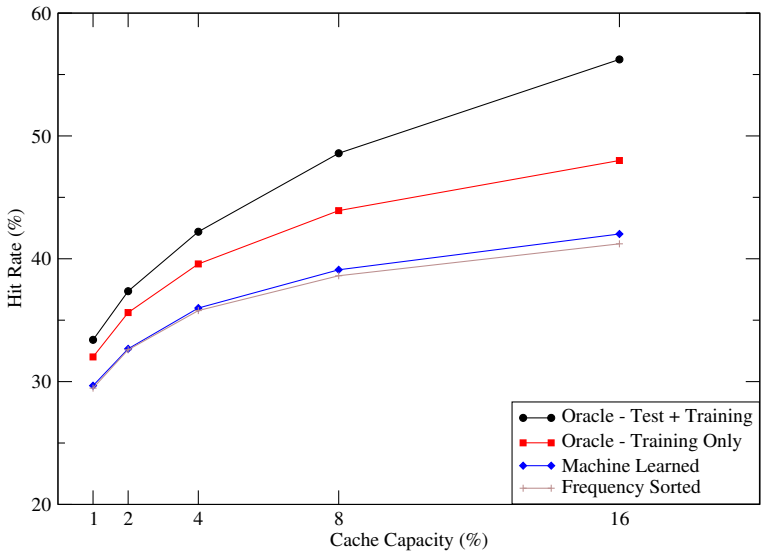


Figure 2.3: Comparison of machine learned static caching strategy versus Oracle static caching strategies and baseline frequency-based strategy.

Figure 2.3 compares the baseline algorithm LFU and the machine learned caching policy with two Oracle algorithms. Although the results seem that, our proposed machine learned caching policy improve the hit rate of the static cache only by 0.66%, the comparison with the optimal methods show that this constitute a significant improvement of 10% within the room of possible improvement. The comparison of the two oracles also hints the difficulty of the caching problem. There is almost 8% difference in hit rate when the queries that only appear in the test set is included in the static

cache. These queries require a "completely clairvoyant" method to be added to the static cache, which is for all practical purposes, impossible. This also hints the robustness of the user behavior and shows it may be possible that the room for improvement might be even tighter than what is presented here.

2.6 Dynamic Caching

In this section, we evaluate the dynamic result caching problem. For this purpose, we take the other extreme case, where the result cache is composed of only the dynamic part. In order to evaluate the room for improvement in the dynamic caching problem, we present Belady's algorithm as an optimal method of dynamic caching. We then present our proposed machine learning approach for the dynamic caching problem and validate its effectiveness.

2.6.1 Techniques

The dynamic caching problem is quite different than static caching. While in static caching it is not possible to exchange queries that are in the cache, dynamic caching allows us to evict queries from the cache in exchange for some other query that would be more beneficial for the time being. In that sense, dynamic caching is more flexible than static caching. In literature (45), the results with the hybrid models show that static

caching is more effective for detecting steady behavior (i.e. frequency) of user queries while dynamic caching is used for elevating the effectiveness of a caching policy by detecting bursty behavior (i.e recency) in user queries.

In order to evaluate the effects of dynamic caching, we conduct experiments on a fully dynamic cache with several caching policies. As a baseline dynamic caching policy we used the LRU caching policy. We also present the results of Belady's algorithm as an optimal dynamic caching policy. Finally, we applied our proposed machine learning approach to dynamic caching and evaluate the results. The algorithms we have used can be summarized as:

Least Recently Used (LRU): This is the underlying caching policy for recency. LRU attempts to fill the cache with the most recent queries.

Belady's Algorithm: The best possible strategy for a cache with finite size would be to always keep the queries that would be referenced last in the future. This optimal caching strategy is referred to as the clairvoyant algorithm or the Belady's algorithm. The impracticality of implementing such a caching strategy in an online framework comes from the fact that it is not possible to know future queries during execution. For representing the room for improvement in the caching problem, we implemented the Belady's algorithm as a method that "knows" the future query references during warmup and test phases.

Machine Learned: In this strategy, we again used our proposed machine learning

approach to the training set. Similar to the machine learned static caching strategy, we first fitted two regression models to all queries, where the first model gives an estimate whether a query is singleton or not and the second model gives an estimation of query's IAT-Next. We then use the multiplication of both regressions and calculate a query quality metric. For dynamic result caching, we keep the queries with the highest quality values in the cache, evicting queries with lower quality values.

During our experiments, we have made an important observation concerning the performance of the machine learned dynamic result cache. When the regression model is used to decide which queries to keep in the dynamic cache, the predicted singleton queries, the singleton queries that are predicted as popular, could pollute the cache and degrade the performance of the result cache severely. The reason of this behavior is that such singleton queries may rank better than some frequent queries, effectively preventing them from getting admitted in the long run.

In order to prevent pollution of the cache due to misclassifications, we propose a segmentation method that merges and honors the LRU policy. According to our segmentation method, we partition the query result cache into a fixed number of segments. After each time we process a fixed number of queries, we start a fresh cache partition that we call a "segment", in order to write the incoming queries. In our method, the old segment become stale since the proposed method quits writing into the old segment. The queries that take hits within the old segment/s are also removed from their respective segments and admitted into the new segment. Additionally, whenever a query

needs to be evicted from the result cache, the eviction decision is performed only on the queries that are in the oldest segment. Note that all segments are governed by the same machine learned policy, and the only difference is that by honoring LRU more, it is possible to evict old, polluting queries from the cache without any other means of interference.

2.6.2 Results

We conduct experiments on a dynamic cache using different cache capacities and segment sizes. These experiments have three motivations. First, we evaluate the effects of varying segment size on the machine learned cache performance. Second, we analyze whether there is a "most suitable" segment size for different cache capacities. Third, we evaluate whether it is possible to find the best segment size for the machine learned cache prior to testing. That is, whether the best segment size found by solely using the training data would perform equally well while testing the performance of the cache or not.

Figure 2.4 shows the performance of the machine learned cache with different cache capacities and segment sizes. The main purpose of this experiment is to find the best segment size for each cache capacity. Thus, the machine learning model used in these experiments is tested the training data to find the most effective segment size prior to testing..

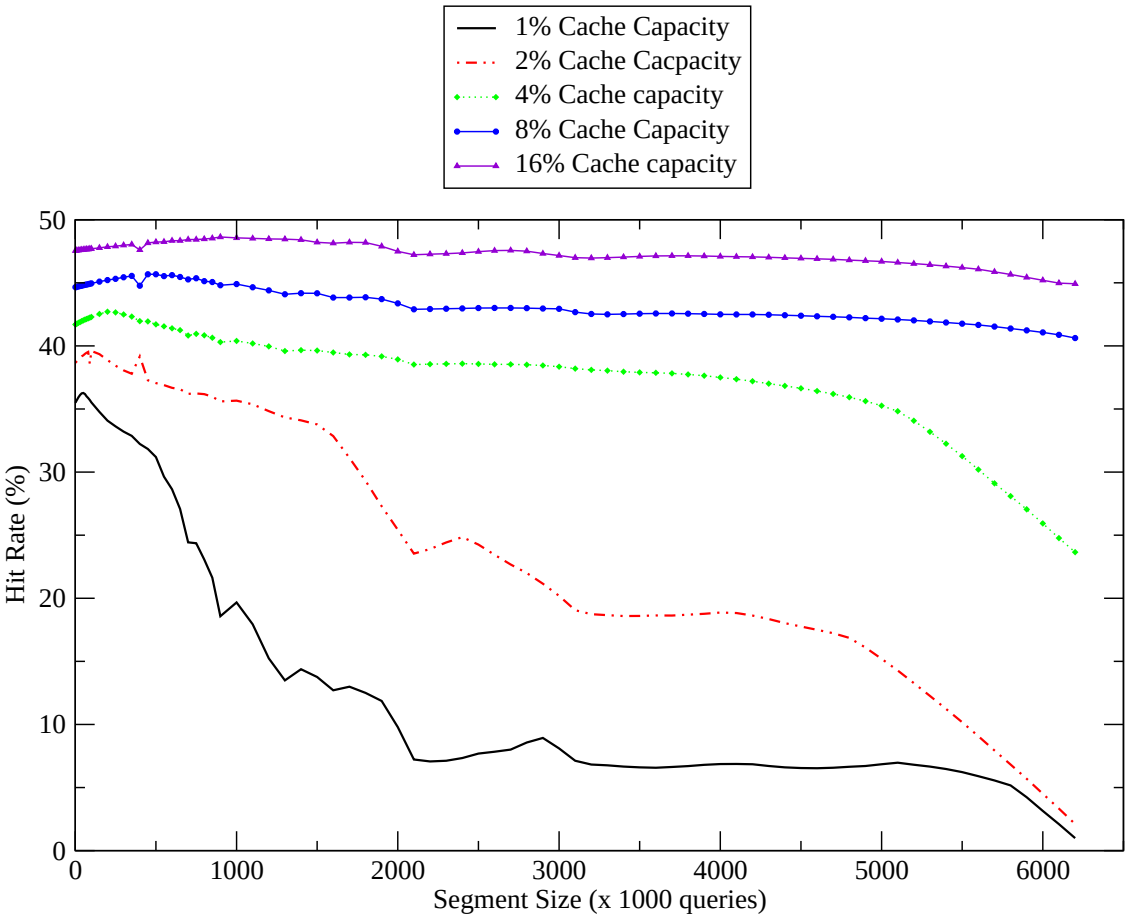


Figure 2.4: Effect of segment size on hit rate. Machine learned dynamic caching policy with varying segment sizes.

For all cases, when the segment size is 1 query, the machine learned caching method performs exactly like LRU policy. This is because, eviction decisions are given over the oldest segment, and due to the order of segments, our approach exhibit a recency-sorted behavior. For the 1% cache capacity, increasing the segment size up to 30,000 queries also increases the hit rate of the caching policy. However, with larger segment sizes, the performance of the algorithm continuously drop down, eventually to 2.7% when using only 1 segment. This is due to the fact that, small segment sizes respect LRU policy more while larger segment sizes respect the machine learned

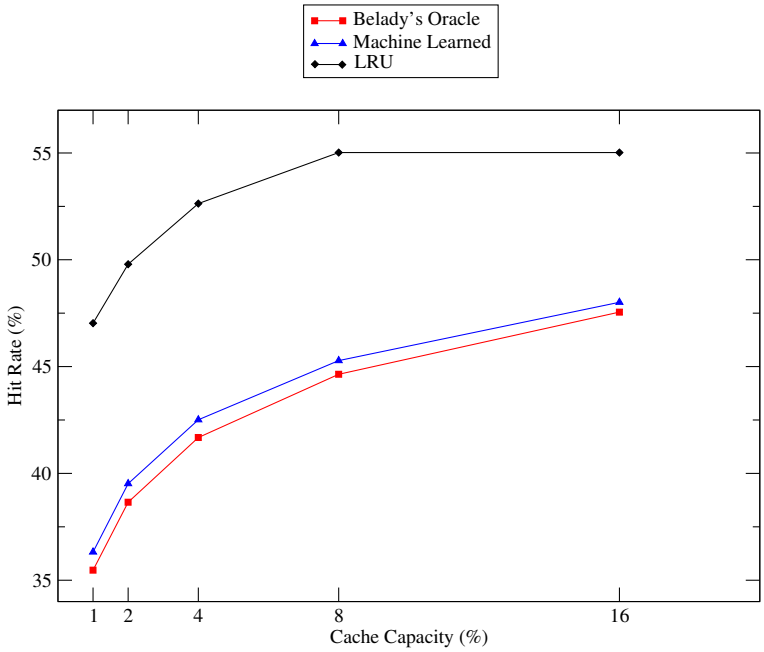


Figure 2.5: Comparison of machine learned caching policy, baseline policy LRU, and Belady's algorithm.

approach more. Increasing the segment size in a small scale benefit the caching policy, since using the machine learning approach enables the policy to give more correct caching decisions. However, further increasing the segment size leads to cache pollution due to misclassification of singleton queries. When the cache capacity is very small, the effects of this cache pollution is much more apparent. Both for cache capacities 1% and 2of the result cache drops down significantly with increasing segment size. However, with larger segment sizes the performance degradation due to cache pollution is almost negligible.

Next, we conducted experiments to compare the effectiveness of the machine learned dynamic caching policy with the optimal caching algorithm and the baseline

LRU. Figure 2.5 shows the results of these experiments. We use the best segment sizes for each cache capacity by using the results from the experiment above. The results show that our proposed machine learned approach performed consistently better than the baseline LRU policy for all cache capacities. For 16% cache capacity, the machine learned caching policy improves LRU by 0.65% which constitutes 7.4% of the possible room for improvement.

Table 2.3 shows the 10 most discriminating features selected by GBDT for singleton prediction model and next arrival time regression model, respectively. In the table, columns 1–3 denote the most discriminating features for singleton prediction, and columns 4–6 denote the most discriminating features for next arrival time prediction. It is notable that, frequency of a query is selected as the most important indicator of the singleton prediction, while features that hint more on popularity of a query, such as Query_Time_Compatibility and Top1_CLICK are selected as best indicators for IAT-Next regression. Another notable feature within these results are that Page_Number being important in singleton prediction while not rated highly for IAT-Next regression.

2.7 Static-Dynamic Caching

In this section, we use the insights we have gathered from our experiments with the two extreme cases in query result caching, the fully static and fully dynamic caching, and combine these two methods in a sensible manner. As a baseline method we selected the

state-of-the-art SDC and apply our machine learning approach on SDC. To this end, we will first explain the different techniques that we used for evaluating the machine learning approach, and next we present experimental results.

2.7.1 Techniques

For evaluating the effectiveness of the machine learning approach for query result caching we implemented several policies. As a baseline method, we implemented the static-dynamic cache (SDC) strategy. We also perform experiments with two different caching policies and propose three optimality bounds for the static-dynamic result caching. The caching policies we evaluated are:

Static-Dynamic Cache (SDC): According to SDC (45), the result cache is divided into 2 segments. A static segment and a dynamic segment. The static segment is created using the most frequent queries in the dataset and the dynamic segment uses a LRU eviction policy. For SDC, the best ratio of this division is found through experimentation and may vary for different datasets depending on the query submission characteristics. For our dataset, our experiments yield the best results when we set the static segment size as 70% of the total result cache.

Belady's Algorithm: The optimal algorithm used in this caching policy is the same as the policy explained in section 2.6.

SDC - Dynamic Oracle: This policy is based on the SDC strategy, where Belady's algorithm is used for admission and eviction decisions on the dynamic part. The static part is created using the most frequent queries in the training set. This policy gives an even tighter bound than the Belady's algorithm for the optimality condition of SDC strategy. The performance of this method gives an estimate for the room of improvement in the dynamic segment of SDC.

SDC - Static Oracle: In this strategy, instead of filling the static segment of SDC with the most frequent queries in the training set, we create the static contents of the result cache using the most referenced queries in the test set. The dynamic segment uses LRU policy for admission and eviction decisions. The performance of this policy would present an insight concerning the room for improvement in the static segment of SDC.

SML + LRU: In this caching policy, we applied our proposed machine learning approach to the static segment of SDC using the method presented in 2.5.

Machine Leaned Static-Dynamic Cache (MLRU): We applied our proposed machine learning approach to both static and dynamic segments of SDC. The training method for both segments are the same techniques presented in sections 2.5 and 2.6 respectively.

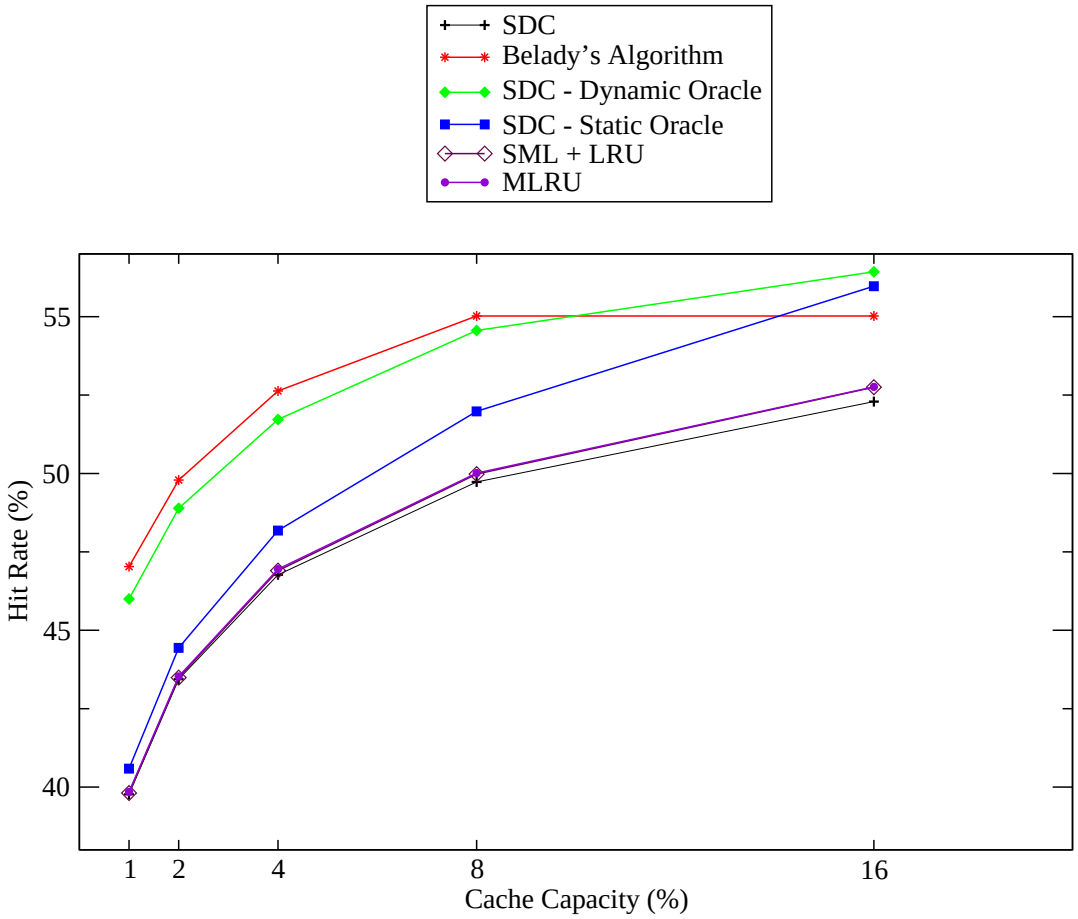


Figure 2.6: Comparison of different result caching policies for various cache sizes.

2.7.2 Results

Figure 2.6 shows the relative performances of different static-dynamic caching policies on varying cache capacities. In this figure, Belady's algorithm gives the optimal caching algorithm when cache is fully dynamic. It can also be used to roughly estimate an optimality condition for static-dynamic caching. SDC-Dynamic and SDC-Static Oracles do not actually show an upper bound for the room of improvement in the static-dynamic caching problem. However, they give an idea of room for improvement in the dynamic and static segments of SDC.

One striking observation that can be made on the results of these experiments is that, although Belady's algorithm is indeed the optimal caching strategy, both SDC-Dynamic and SDC-Static outperform it on large cache sizes. This is due to the fact that Belady's algorithm is optimal only for dynamic caching and is prone to non-compulsory misses (misses that happen when a query is seen for the first time), while static segments of the two latter algorithms are not since static segments are pre-computed and placed in the cache ahead of time. Thus, it is possible for a static approach to outperform the dynamic optimal.

Comparison of the two machine learning approaches with SDC show that, both machine learned caching policies outperform SDC for all cache capacities. Also, the machine learned caching method perform better at larger cache capacities. For 16% cache capacity, machine learned caching policy improves the performance of SDC by 0.47%, which is more than 11% of the possible improvement against the best oracle. When two machine learned approaches are compared with each other, both machine learned algorithms perform almost equally. Addition of the dynamic machine learned strategy does not seem to benefit the static-dynamic case as large as expected, and although MLRU performs better than SML+LRU, the improvement is almost negligible.

Table 2.1: The features used in our machine learning approach

Abbreviation	Feature Description	Feature Category
Q_LEN	Query_Length	
WORD_C	Word_Count	
URL_PRESENT	Is_URL_Present	Query String-Based
DOM_PRESENT	Is_Domain_Present	Features
SPELL_CORR	Is_Spell_Corrected	
AVG_QT_LEN	Average_Query_Term_Length	
USER_LOGGED	Is_User_Logged	
PAGE_NUM	Page_Number	User-Based
CLICK_C	Click_Count	Features
TOP1_CLICK	First_Link_Click_Count	
HIT_C	Total_Number_Of_Hits	
RAREST_TERM	Rarest_Query_Term_Index_Size	Search Engine
COMMON_TERM	Most_Common_Query_Term_Index_Size	Related Features
AVG_TERM	Average_Index_Size	
MIN_TFREQ_MIN	Minimum_Term_Frequency_Last_Minute	
MAX_TFREQ_MIN	Maximum_Term_Frequency_Last_Minute	
AVG_TFREQ_MIN	Average_Term_Frequency_Last_Minute	
MIN_TFREQ_HOUR	Minimum_Term_Frequency_Last_Hour	Term Frequency-
MAX_TFREQ_HOUR	Maximum_Term_Frequency_Last_Hour	Based Features
AVG_TFREQ_HOUR	Average_Term_Frequency_Last_Hour	
MIN_TFREQ_DAY	Minimum_Term_Frequency_Last_Day	
MAX_TFREQ_DAY	Maximum_Term_Frequency_Last_Day	
AVG_TFREQ_DAY	Average_Term_Frequency_Last_Day	
QFREQ	Overall_Query_Frequency	
QFREQ_MIN	Query_Frequency_Last_Minute	Query Frequency
QFREQ_HOUR	Query_Frequency_Last_Hour	Features
QFREQ_DAY	Query_Frequency_Last_Day	
Q_TIME	Query_Submission_Hour	
DAY_C	Query_Day_Count	Temporal Features
TIME_COMP	Query_Time_Compatibility	

Table 2.2: The most discriminating 10 features for machine learned static caching strategy

Rank	Feature	Feature Importance Rate
1	QFREQ	100
2	TIME_COMP	35.3822
3	QFREQ_DAY	34.7971
4	WORD_C	17.8072
5	RAREST_TERM	17.1516
6	PAGE_NUM	11.9368
7	QFREQ_HOUR	11.9225
8	Q_LEN	11.2414
9	CLICK_C	9.6561
10	USER_LOGGED	8.8371

Table 2.3: The most discriminating 10 features for machine learned dynamic caching strategy.

Singleton Prediction			Next Arrival Time Prediction		
Rank	Feature	Feature Importance Rate	Rank	Feature	Feature Importance Rate
1	QFREQ	100	1	TIME_COMP	100
2	TIME_COMP	41.1229	2	TOP1_CLICK	49.1376
3	WORD_C	17.2646	3	MIN_TFREQ_HOUR	43.2057
4	USER_LOGGED	14.8581	4	WORD_C	42.1965
5	PAGE_NUM	10.8796	5	QFREQ	41.1560
6	CLICK_C	10.2059	6	USER_LOGGED	33.1946
7	Q_LEN	9.1032	7	HIT_C	30.6932
8	TOP1_CLICK	9.0213	8	CLICK_C	27.5653
9	MIN_TFREQ_DAY	8.5213	9	MIN_TFREQ_DAY	26.9315
10	HIT_C	8.2852	10	SPELL_CORR	22.8226

Table 2.4: The most discriminating 10 features for machine learned SDC caching strategy for cache capacities 1% and 16%.

1% Cache Capacity					
Singleton Prediction			Next Arrival Time Prediction		
Rank	Feature	Feature Importance Rate	Rank	Feature	Feature Importance Rate
1	QFREQ	100	1	QFREQ_DAY	100
2	QFREQ_DAY	62.1077	2	WORD_C	31.7027
3	QFREQ_HOUR	24/8774	3	QFREQ	22.6615
4	WORD_C	23.0789	4	MIN_TFREQ_DAY	21.4185
5	RAREST_TERM	19.8314	5	RAREST_TERM	20.1371
6	PAGE_NUM	16.1379	6	QFREQ_HOUR	18.8391
7	AVG_TERM	8.6129	7	SPELL_CORR	15.8205
8	QFREQ_DAY	6.3245	8	AVG_TERM	13.3344
9	SPELL_CORR	4.9722	9	AVG_TFREQ_DAY	11.0508
10	MIN_TFREQ_MIN	2.7914	10	PAGE_NUM	9.4277

16% Cache Capacity					
Singleton Prediction			Next Arrival Time Prediction		
Rank	Feature	Feature Importance Rate	Rank	Feature	Feature Importance Rate
1	QFREQ	100	1	QFREQ_DAY	100
2	QFREQ_DAY	37.1112	2	WORD_C	29.6291
3	TIME_COMP	35.7186	3	QFREQ	21.9625
4	WORD_C	19.4112	4	HIT_C	20.6426
5	RAREST_TERM	17.0336	5	RAREST_TERM	19.6790
6	QFREQ_HOUR	14.9345	6	TIME_COMP	19.6191
7	PAGE_NUM	14.4185	7	QFREQ_HOUR	18.2379
8	Q_LEN	9.3008	8	SPELL_CORR	15.5671
9	DAY_C	7.6380	9	AVG_TERM	13.1281
10	AVG_TERM	5.7492	10	DAY_C	11.0828

Table 2.4 show the 10 most discriminating features for two different cache capacities, 1% and 16%, for both singleton and IAT-Next prediction. In the table, columns 1–3 denote the most discriminating features for singleton prediction, and columns 4–6 denote the most discriminating features for IAT-Next prediction. The first 10 rows denote the feature importances for 1% cache capacity and the last 10 rows denote the feature importances for 16% cache capacity. First observation that can be done on Table 2.4 is that the most discriminating features for both Singleton prediction and IAT-Next prediction are very similar apart from several rank shifts for both cache capacities. When compared to the results presented in Table 2.3, addition of the static cache to the dynamic problem seem to affect the IAT-Next prediction adversely, degrading the regression model towards the singleton prediction model.

Although when the cache capacity grows larger the improvement rate of the machine learned caching policy increases, the improvement fail to meet the expectations that can be inferred from both fully static and fully dynamic caching experiments. There can be two different explanations for this behavior. It is either the frequency features start to lose their importance, or temporal features, such as Query_Time_Compatibility and Query_Day_Count start to gain importance. If the former case is true, then we can conclude that the dynamic models start to suffer from overfitting, which is a common problem in decision tree learning. However if the latter case is true, then we can conclude that with the growing cache capacity, the machine learning algorithm start to utilize other features and start to perform better. Although

we indicate both of these two possibilities, since features such as Page_Number and Word_Count still have similar feature importance rates with respect to query frequency in both cache capacities, we strongly believe that the latter case is true.

2.8 Discussions

The query distribution of the query log is essential for understanding the nature of the caching problem, the relationship between static and dynamic caching and the effectiveness of the SDC policy. In this work, the query frequency distribution of the examined search engine query log follows a power law distribution (164). In literature, power law graphs, occording to their frequency distribution, can be partitioned into 3 segments (27) for sake of data analysis. These segments are called: The head, torso, and tail. The head of the query log contains the most frequent queries, which also represent a large portion of the search engine query traffic. The queries in the tail on the other hand, appear in the query log very rarely, which represent unpopular and unanticipated queries.

The SDC policy partitions the cache exploiting these three segments. By storing the head of the query log in the static segment, SDC policy is able to respond to a large portion of the incoming queries from the cache without the need of any dynamic caching policy. The rest of the cache space is reserved as a dynamic cache, responding to the torso and the tail of the query log, in the hope of detecting and responding to

rather infrequent queries. Although, the motivation of SDC puts a great emphasis on the static portion of the cache for responding to frequency of queries, in both (98) and (45) it is shown that a static cache alone does not perform well for real life querie sets. In other words, for an effective caching policy, caching recent but rather less popular queries is almost as important as caching the frequent queries.

During our experimentation we came up with similar conclusions. Some of the features that we initially predicted as potentially very discriminative and influential for differentiating turn out to rank low at feature importance order for both singleton prediction and IAT-Next regression. As an illustrative example, we anticipated that Page_Number feature would be an important feature for distinguishing between frequent and infrequent queries. However, the feature importance values in our experiments show that this feature bear little value for a query being frequent or not. Our first hand observations over the data lead us to the following hypothesis: some automated systems, such as Web bots and crawlers are continuously submitting queries with large page numbers, leading to unanticipated feature values. In fact, through observation, we have also verified that there are several robot query submission activities within our dataset. However, since we have no empirical method to identify or verify that a query is definitely the result of some automated activity, we are unable to provide these results here.

The application of the proposed machine learning approach to the static-dynamic caching also introduce several difficulties. Although distinguishing the head segment

from the torso segment is rather a trivial problem for our approach, our machine learning models have difficulties for distinguishing torso queries from the tail queries. Our experiments show that, for machine learned SDC caching policy, the improvements are mostly gained via the machine learned static cache segment, and the benefit of machine learned dynamic segment on top of the improvement gain from the static segment is very small.

Introduction of the static cache to the caching problem also results in a harder dynamic caching problem. Our experimental results show that, for our dataset, using an SDC cache with 1% cache capacity where 70% of the cache dedicated to the static cache, all queries with frequencies higher than 136 would be stored in the static cache. For an SDC cache with 16% cache capacity and 70% static cache, the queries with frequencies higher than only 4 are stored in the static cache. Using static cache segment, where the cache size is considerably large leaves dynamic cache segment with only a small, but harder portion of the problem where all queries are in the torso or in the tail portion of the query log. The difficulty of dynamic caching problem with a static component comes from the fact that, tail and torso queries contain very little distinguishable information. Our conclusion is that, the features we have used and that are proposed in literature are not, without other inference mechanisms, such as storing additional temporal information or using some other means of information for inferring query popularity, adequate and does not contain enough distinguishable information for addressing the dynamic caching problem for the current real-life query logs. That

is mostly because the user and query behavior in today's search engines is much more dynamic and unanticipated than query logs examined in the literature in the past.

Finally, our experiments also show a potential problem in using machine learning techniques for finding a dynamic caching policy. Due to misclassifications during testing, the use of machine learning methods cause a cache pollution, where the misclassified queries start to occupy cache space. Due to the over-valuation of such queries, the machine learned policy can tend to evict more valuable queries. In this work, we propose a cache segmentation method to alleviate this problem. In our approach, after processing a constant number of queries, the dynamic segment of the cache is re-started so that the misclassified instances can be evicted from the cache. Although, this method prevents pollution to a degree, more accurate machine learning methods is ultimately required for a better caching policy.

Chapter 3

Chat Mining:

Predicting User and Message

Attributes in

Computer-Mediated Communication

3.1 Introduction

With the ever-increasing use of the Internet, computer-mediated communication via textual messaging has become popular. This type of electronic discourse is observed in point-to-point or multicast, text-based online messaging services such as chat servers,

discussion forums, emails and messaging services, newsgroups, and IRCs (Internet relay chat). These services constantly generate large amounts of textual data, providing interesting research opportunities for mining such data. We believe that extracting useful information from this kind of messages/conversations can be an important step towards improving the human–computer interaction.

According to a study by (71), "electronic discourse is neither writing nor speech, but rather written speech or spoken writing, or something unique." Due to its mostly informal nature, electronic discourse has major syntactic differences from discourse in literary texts (e.g., word frequencies, use of punctuation marks, word orderings, intentional typos). The informal nature of electronic discourse makes the information obtained more realistic and reflects the author attributes more accurately. Analysis of electronic discourse may provide clues about the attributes of the author of a discourse and the attributes of the discourse itself.

Specifically, machine learning can be a powerful tool for analyzing electronic discourse data. This work particularly concentrates on the data obtained from chat servers, which provide a point-to-point online instant messaging facility over the Internet. We investigate the rate of success in the problem of predicting various author- and message-specific attributes in chat environments using machine learning techniques. For this purpose, we first employ a term-based approach and formulate the chat mining problem as an automated text classification problem, in which the words occurring in chat messages are used to predict the attributes of the authors (e.g., age, gender) or

the messages (e.g., the time of a message). Second, we employ a style-based approach and investigate the effect of stylistic features (e.g., word lengths, use of punctuation marks) on prediction accuracies, again for both author and message attributes. Finally, we briefly discuss the effect of the author and message attributes on the writing style.

The main contributions of this study are four-fold. First, the chat dataset used in this work has unique properties: the messages are communicated between two users; they are unedited; and they are written spontaneously. We believe that extracting information from real-time, peer-to-peer, computerized messages may have a crucial impact on the areas such as financial forensics, threat analysis, and detection of terrorist activities in the near future. Our work presents a new effort in that direction, aiming to retrieve previously unexplored information from computerized communications. Second, for the first time in the literature, several interesting attributes of text and its authors are examined. Examples of these attributes are educational affiliations and connectivity domains of the authors and the receivers of the messages. Third, the performance of term- and style-based feature sets in predicting the author and message attributes are compared via extensive experimentation. Fourth, to the best of our knowledge, our work is the first one that investigates real-time, peer-to-peer, computerized communications in the context of authorship studies. Our findings are good pointers for researchers in this new application area, namely chat mining.

The rest of this chapter is organized as follows. Table 3.1 displays a list of frequently used abbreviations in this work. We provide a detailed literature survey of the

Table 3.1: The summary of abbreviations

AA	Authorship attribution	***k*-NN**	K-nearest neighbor
AC	Authorship characterization	**NB**	Naive Bayesian
CE	Cross entropy	**NN**	Neural networks
DA	Discriminant analysis	**PCA**	Principal component analysis
DT	Decision trees	**PRIM**	Patient rule induction method
EG	Exponentiated gradient	**RM**	Regression models
GA	Genetic algorithms	**SD**	Similarity detection
HMM	Hidden Markov models	**SVM**	Support vector machines
IRC	Internet relay chat	**TC**	Text classification

related work in Section 3.2. In Section 3.3, we discuss the characteristics of computer-mediated communication environments and elaborate on the information that can be extracted from such environments. Section 3.4 introduces the chat mining problem and discusses our formulations, which are based on the use of term- and style-based feature sets. In Section 3.5, we provide information about the dataset used in this study and present our framework for solving the chat mining problem. Section 3.6 provides the results of a large number of experiments conducted to evaluate the feasibility of predicting various author and message attributes in a chat environment. In Section 3.7, we finalize the chapter with a concluding discussion.

3.2 Related Work

In the last ten years, the Internet has become the most popular communication medium. Chat servers, IRCs, and instant messaging services provide online users the ability to communicate with each other simultaneously. Discussion forums, emails, and newsgroups enable their users to create virtual communities regardless of geographical and political barriers. This information dissemination platform provides new research possibilities such as assessing the task-related dimensions of the Internet use. In their work, (42) examine the communication process of chat users in an industrial setting. They investigate how customers and customer service representatives respond to each other and identify the reasons of miscommunication between partners. The collaborative work within virtual groups is explored by (151). The authors identify six communication rules for enhancing trust, which in turn enable chat users to work more efficiently. (118) examines several problems concerning communications in a virtual library reference service. The quality of chat encounters between librarians and clients, compensation of lack of emotional cues, and relational dimensions of chat references are among the questions investigated. The author identifies several relational facilitators, communication themes and concludes that computer-mediated communication is no less personal than face-to-face communication.

Understanding the user behavior is another aspect of the ongoing research on computer-mediated communication. (119) examine the communication and information seeking preferences of the Internet users. They also compare traditional libraries and the Internet as the means for an information repository and emphasize the fact that the Internet is starting to become an alternative for text-based communication.

The investigation of chat user attributes is another dimension that attracts researchers. (60) examine gender variations in Web logs using logistic regression techniques. However, the authors cannot find any conclusive results binding the users' genders and Web writings. In their work, (61) examine several aspects of the language use in the Internet. They assert that gender is reflected in online discourse in every language they studied.

Extracting interesting information from anonymous electronic document collections using authorship attribution may also provide several research opportunities. A quick literature survey reveals the fact that the previous studies in authorship attribution were mostly considered in the context of law enforcement (145), religious studies (117; 124), and humanities (33; 44; 109). In the past few years, the examination of electronic discourse in the context of authorship studies started to got attention of a growing number of researchers.

The history of authorship studies dates back to more than two millennia. The first work in literature is reported in the fourth century BC, when the librarians in the

famous library of Alexandria studied the authentication of texts attributed to Homer (96). Since then, a large number of documents have been the focus of authorship studies. Broadly, the authorship studies in literature can be divided into three categories (40; 163): authorship attribution, similarity detection, and authorship characterization.

Authorship attribution is the task of finding or validating the author of a document. Some well-known examples of authorship attribution are the examination of Shakespeare's works (44; 65; 105) and the identification of the authors of the disputed Federalist Papers (64; 92; 109; 146). Similarity detection aims to find the variations in the writing style of an author (114) or to find the resemblances between the writings of different authors, mostly for the purpose of detecting plagiarism (55).

Authorship characterization is the task of assigning the writings of an author into a set of categories according to the author's sociolinguistic attributes. Some attributes previously investigated in literature are gender (77; 81; 149), language background (149), and education level (72). (77) and (81) evaluated methods for determining the gender of a document's author. (149), in addition to gender, tried to predict the language background of authors using machine learning techniques. (72) analyzed the educational backgrounds of the authors employing cross entropy.

With the advent of computers, it has become possible to employ sophisticated

techniques in authorship analysis. The techniques employed in authorship analysis can be broadly categorized as statistical and machine learning techniques. Examples of statistical techniques are Hidden Markov models (75), regression models (74), cross entropy (72), discriminant analysis (33; 73; 79; 141), and principle component analysis (10; 28; 63). Machine learning techniques are also frequently used in authorship studies. Most commonly used techniques are *k*-nearest neighbor (79; 81; 138), naive Bayesian (76; 81; 138), support vector machines (39; 40; 70; 144; 163), genetic algorithms (64), decision trees (163), and neural networks (55; 76; 105; 79; 81; 138; 146; 163).

With the widespread use of computers, new pursuits that reflect the personal characteristics of individuals drew attention of authorship studies. Computer programming and musical composition are examples of such pursuits. (79; 136) used several structural and syntactic features to predict the author of a program. They generate these features by analyzing the variations in programming construct preferences of the authors. The work of (136) achieved 73% accuracy in predicting the author of 88 programs written by 29 different authors. In their work, (11) analyzed the musical style of five well-known composers using various classification algorithms on a dataset with computer-generated features like the stability measures of the composition, voice density, and entropy measures.

The emergence of electronic discourse also presents interesting opportunities for

authorship analysis. As electronic discourse becomes a popular form of communication, detecting illegal activities by mining electronic discourse turns out to be important. In their work, (149) analyzed the information in email messages in order to identify the distinguishing features in writing styles of emails for predicting authors' identity, gender, and language background. In addition to some well-known stylistic features, they used features like smileys and emoticons. They achieved 72.1% and 85.6% accuracies in predicting the gender and language background of more than 300 authors, respectively.

(144) analyzed email messages for predicting the identity of their authors using a term-based feature set. (141) analyzed the gender of a number of email authors and concluded that email authors had used gender-preferential language in informal electronic discourse. (163) constructed a language-independent framework to predict the identity of the author of online Chinese and English newsgroup messages. For a selection of 20 authors they have succeeded in predicting the identity of the authors with an impressive 95% accuracy for the English message collection and 88% accuracy for the Chinese message collection. (7) also studied newsgroup messages for identification of the authors using a style-based classification approach. Although they used a highly imbalanced dataset, over 40% accuracy is achieved in predicting the messages of 20 different authors.

(163) presented a table that provides a summary (features used, type of analysis,

Table 3.2: A summary of the previous works on authorship analysis

Study	Type	Technique	Features
(109)	AA	Statistics	style
(28)	AA	PCA	style
(44)	AA	Statistics	both
(73)	TC	DA	style
(76)	AA	NB, NN	style
(105)	AA	NN	style
(64)	AA	GA	style
(10)	AA	PCA	both
(74)	TC	RM	style
(79)	AA	*k*-NN, DA, NN	style
(70)	TC	SVM	term
(138)	AA, TC	*k*-NN, NB, NN	style
(75)	AA	HMM	term
(141)	AC	DA	style
(144)	AA, AC	SVM	style
(149)	AC	SVM	term
(7)	AA	EG	style
(40)	AA	SVM	style
(114)	AC	DA	style
(55)	SD	NN	style
(72)	AA, AC	CE	term
(65)	AC	SVM	style
(81)	AC	*k*-NN,NB, NN	both
(163)	AA	SVM, DT, NN	style

and dataset properties) of the previous works on authorship analysis. Here, we provide a similar table with additional information for a number of previous works. In chronological order, Table 3.2 gives details such as the analysis techniques used in the works and the type of the features used (i.e., term-based or style-based features). In compliance with our previous taxonomy, the table categorizes each work as an authorship attribution (AA), similarity detection (SD), or authorship characterization (AC) task. Several text classification (TC) works, which are closely related with authorship studies, are also displayed in the table.

3.3 Computer-Mediated Communication

3.3.1 Characteristics

Using textual messages in order to interact with other people is a popular method in computer-mediated communication. Point-to-point instant messaging, also referred to here as chatting, has several properties which makes it unique with respect to both literary writing and messaging in other types of online services: Messages (1) are written by users with a virtual identity; (2) specifically target a single individual; (3) are unedited; and (4) have a unique style and vocabulary. Below, we elaborate more on these characteristics.

In most chat servers, the real identity of a user is hidden from other users by a virtual identity, called "nickname." Typically, the users have the option of building up this virtual identity and setting its different characteristic features. This gives the users the opportunity to provide others false information about their real identities. For example, a male user may set the gender of his virtual identity as female and try to adapt his writing style accordingly to fool others. Having such misleading information in chat environments makes authorship attribution and characterization quite difficult even for domain experts.

Unlike literary writing, where the documents are written for public audience, chat messages target a particular individual. Most often, chat messages are transmitted

between two users, that is, each message has a specific sender and a receiver. The writing style of a user not only varies with his personal traits, but also heavily depends on the identity of the receiver. For example, a student may send a message to another student in a style which is quite different from the style of a message he/she writes to his supervisor. This type of an ability of effectively changing one's writing style is known as sociolinguistic awareness (56). As an interesting genre detection task, chat messages can be examined in order to find out who the receiver is.

Books and plays are the most common type of literary material used in authorship analysis (47). This type of documents are usually modified by editors who polish the initial drafts written by authors. Hence, most of the time, the writing style of the original author is mixed with that of an editor. (122) discusses the undesirable effects of this type of editing on authorship analysis and concludes that edited texts are hard to mine since stylistic traces of the author and the editor are not separable. The real-time nature of chat messages prevents any editorial changes in electronic discourse, and thus the writing style reflects that of the original author. In this aspect, it is quite valuable to work on unedited chat messages. However, in the mean time, having no editorial modifications means that, in chat messages, misspellings are more frequent compared to edited text. It is debatable whether these misspellings are part of an author's writing style or not.

Due to its simultaneous nature, electronic discourse reflects the author's current emotional state much better than any other writing. Since the messages transmitted

between users are purely textual, chat messaging has evolved its own means for transferring emotions. Emoticons (emotion icons) are commonly known and widely used ways of representing feelings within computer-mediated text (152). We restrict our work on a particular subset of emoticons: smileys. Smileys, (e.g, ":-)" and ":-(") are sequences of punctuation marks that represent feelings such as happiness, enthusiasm, anger, and depression. Repetition of specific characters in a word can also be used as a means of transferring emotions by putting an emphasis on a text. (e.g. "Awesomeeee!"). In chat messages, the use of such consciously done misspellings is also frequent. Since the use of smileys and emphasized words is highly dependent on the writing style of an author, they pose valuable information. However, preserving such information makes traditional text processing methods (e.g., stemming and part of speech tagging) unsuitable for mining chat messages.

3.3.2 Predictable Attributes

In general, chat messages can be used to predict two different types of attributes: user- or message-specific attributes. In the first type, the distinguishing features of a chat message may be used to predict the biological, social, and psychological attributes of the author who wrote the message. In the latter, the distinguishing features may be used to predict the attributes of the message itself.

Examples of user-specific attributes are gender, age, educational background, income, linguistic background, nationality, profession, psychological status, and race. In this work, we concentrate on four different user-specific attributes: gender, age, educational environment, and Internet connection domain of the users. Among these attributes, the gender of an author is widely examined in literature (81; 149), and it is observed that authors have the habit of selecting gender-preferential words (141). In this work, we also try to predict the user age based on the fact that every generation has its own unique vocabulary. Predicting the age of a user may be useful for profiling the user and hence may help in forensic investigations. Educational environment is also worth studying since it is possible that the vocabulary and writing style of a user might be affected by the school he/she is affiliated with. In order to test this claim, we analyzed the chat messages of users in different universities. We also noted that computer-mediated communication adds new dimensions whose analysis may yield valuable information. As an illustrative task, we try to predict the Internet connection domains of users, which may have veiled means for the educational and occupational status of a user. For example, a user connected from the ".edu" domain probably has an affiliation with a university, whereas a user connected from the ".gov" domain possibly works for the government.

For message-specific attributes, we concentrate on three attributes: author, receiver, and time of the messages. The identity of the author of a given text is the most frequently studied attribute in authorship analysis (64; 79; 109; 163). In case of chat

Table 3.3: The attributes predicted in this work and the number of classes available for each attribute

User-specific attributes	# of classes	Message-specific attributes	# of classes
Gender	2	Receiver	1165
Age	17	Author identity	1616
School	60	Day period	4
Connection domain	7	–	–

mining, the characteristics of chat messages are firmly attached to the author's linguistic preferences. Hence, we try to predict the authors of chat messages as a typical authorship attribution task. The audience of a chat message may also affect the lingual preferences of an author. For the first time in literature, we try to predict the audience of textual documents; i.e., the receivers of the chat messages. The real time nature of chat messages makes it possible to examine whether the time a message is written is predictable. For example, in active hours of the day (morning and afternoon), people may compose long and complex sentences although, in passive hours (nighttime), people may tend to create short and simple sentences. Hence, in this work, we also investigate the predictability of the period of the day a chat message is written.

Table 3.3 presents a complete list of the attributes we try to predict in this work. In this table, the number of classes refers to the maximum number of possible values an attribute can have. For example, the gender attribute has two possible class values (male and female) while the connection domain attribute has seven possible class values, each of which represents a different Internet connectivity domain.

3.4 Chat Mining Problem

The chat mining problem can be considered as a single-label classification problem. If the attribute to be predicted is user-specific, a supervised learning solution to this problem is to generate a prediction function, which will map each user instance onto one of the attribute classes. The prediction function can be learned by training supervised classification algorithms over a representative set of user instances whose attributes are known. In case of message-specific attributes, the process is similar. However, this time, the individual chat messages are the instances whose attributes are to be predicted, and the training is performed over a set of chat messages whose attributes are known.

In predicting the user-specific attributes, each user instance is represented by a set of features extracted from the messages that are generated by that particular user. Similarly, in predicting message-specific attributes, each message instance is represented by a set of features extracted from the message itself. In this work, for predicting both types of attributes, we evaluate two competing types of feature sets: term-based features versus style-based features.

When term-based features are used, the vocabulary of the message collection forms the feature set, i.e., each term corresponds to a feature. In predicting user-specific attributes, the set of terms typed by a user represents a user instance to be classified. In predicting message-specific attributes, the terms in a message represent a message

instance. This type of a formulation reduces the chat mining problem to a standard text classification problem (131).

In literature, term-based feature sets are widely used (86). Unfortunately, term-based features may not always reflect the characteristics of an author since the terms in a document are heavily dependent on the topic of the document. In chat mining, a feature set that is independent of the message topic may lead to better results in predicting the user- and message-specific attributes. Hence, using the stylistic preferences instead of the vocabulary emerges as a viable alternative.

(122) states that there are more than 1000 different stylistic features that can be used to define the literary style of an author. The most commonly used stylistic features are word frequencies; sentence and word lengths; and the use of syllables, punctuation marks, and function words (62). So far, there is no consensus on the set of the most representative features.

This study, in addition to the traditional stylistic features, considers several new and problem-specific stylistic features (e.g., smileys and emoticons) used in order to find better representations for user or message instances. The smileys and emoticons are two important features that are frequently found in chat messages. A summary of the style-based features used in this study is given in Table 3.4. The stylistic features used in this work are grouped into 10 categories. Each category contains one or more features with categorical feature values. For example, the average word length feature

Table 3.4: The stylistic features used in the experiments

Feature category	Features in the category	Possible feature values
character usage	frequency of each character	low, medium, high
message length	average message length	short, average, long
word length	average word length	short, average, long
punctuation usage	frequency of punctuation marks	low, medium, high
punctuation marks	a list of 37 punctuation marks	exists, not exists
stopword usage	frequency of stopwords	low, medium, high
stopwords	a list of 78 stopwords	exists, not exists
smiley usage	frequency of smileys	low, medium, high
smileys	a list of 79 smileys	exists, not exists
vocabulary richness	number of distinct words	poor, average, rich

can possibly have three values: short, medium, and long. This discretization is performed depending on the feature value distributions over a set of messages randomly selected from the chat dataset.

3.5 Dataset and Classification Framework

3.5.1 Dataset

The chat dataset used in this chapter is obtained from a currently inactive chat server called Heaven BBS, where users had peer-to-peer communication via textual messages. The outgoing chat messages (typed in Turkish) of 1616 unique users is logged for a one-month period in order to generate the dataset. The messages are logged without the notice of the users, but respecting the anonymity of messages. The vocabulary of the dataset contains 165 137 distinct words. There are 218 742 chat messages, which

are usually very short (6.2 words per message on the average). The message log of a typical user contains around 160 chat messages.

The dataset also contains users' subscription information such as the name, gender, email address, and occupation. Some fields of the subscription information may be missing as they are optionally supplied by the users. Also, against our best efforts to validate the correctness of the entries, there may be fakes or duplicates among the users.

3.5.2 Classification Framework

In this section, we provide an overview of the framework we developed for solving the chat mining problem. Here, we restrict our framework to prediction of user-specific attributes using the term-based feature set. Extensions of this framework to the message-specific attributes and the style-based feature set are discussed later in this section. Figure 3.1 summarizes the classification procedure used in predicting the user-specific attributes. The framework consists of three stages: data acquisition, preprocessing, and classification. The last two stages contain several software modules that execute in a pipelined fashion.

The corpus creation module of the data acquisition stage forms a tagged corpus from the raw message logs obtained from the chat server. In Figure 3.2, we provide a sample fragment from this corpus. For each user instance in the corpus, between

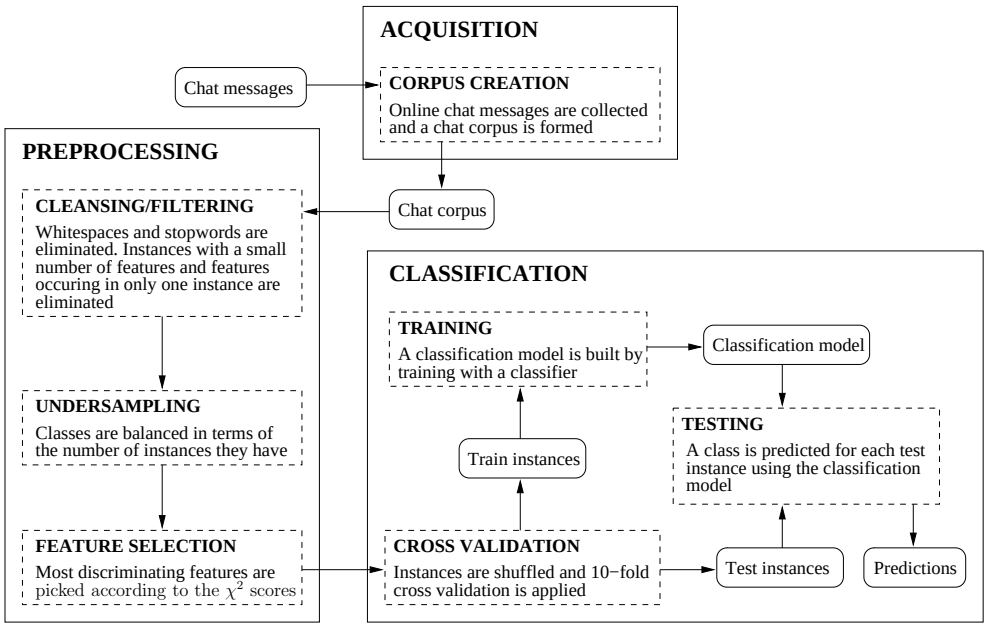


Figure 3.1: The classification framework.

an “INSTANCE” tag pair, the attributes of the user and the messages typed by the user are stored. The target users receiving the messages of the user are separated by the “RECEIVER” tag pairs. Each receiver may receive multiple messages, which are separated by the “X” tag pairs.

After the chat corpus is generated, it undergoes several preprocessing steps to improve classification accuracies. Each preprocessing step is designed as a separate software module. In our framework, the preprocessing stage involves three modules: cleansing/filtering, undersampling, and feature selection.

The cleansing/filtering module aims to obtain a set of representative terms for each user. For this purpose, non-alphanumeric characters (e.g., whitespaces, punctuation marks) are eliminated. A list of 78 Turkish stopwords (i.e., connectives, conjunctions,

```
<INSTANCE=ali>
<NAME=ali guney>
<GENDER=male>
<EMAIL=Guney@alpha.eng.ege.edu.tr>
<DOMAIN=edu>
<SCHOOL=ege>
<BIRTHDAY=19>
<BIRTHMONTH=October>
<BIRTHYEAR=1979>
<HOROSCOPE=libra>
<RECEIVER=blandinka>
<X>
<DATE=Wed Apr 5 16:09:40 2000>
MESELA COK GENIS BIR INSANSIN AMA BAZEN COK KUCUK BIR SEYE
TAKIYOSUN GIBI
//For Example, you are a flexible person.  But Sometimes you concentrate
on small things
</X>
</RECEIVER>
<RECEIVER=ageofeye>
<X>
<DATE=Wed Apr 5 16:10:48 2000>
KONUSMAK ISTIYORMUSUN BENLE
// do you want to talk with me
</X>
</RECEIVER>
</INSTANCE>
```

Figure 3.2: A sample fragment of the chat corpus formed. The user name is deidentified to preserve the anonymity. English translations are added for convenience.

and prepositions) is further used to eliminate content-independent terms. Single-word messages are also ignored since these are mostly uninformative salutations. The features of the user instances are formed by the remaining terms, where the tf-idf (term frequency-inverse document frequency) values (127) are used as the feature values. Finally, the user instances that contain only a small number of features, i.e, those that have less than a pre-determined number of terms, are eliminated.

The existence of imbalanced classes is a crucial problem in text classification (80). If the number of instances selected from each class are not roughly equal, the classifiers may be biased, favoring more populated classes. The main goal of the undersampling module is to balance the number of instances in each class. For this purpose, an equal number of instances with the highest term counts are selected from each class and the remaining instances are discarded. In this dataset, an imbalance is also observed on instance sizes since the number of distinct terms of each user greatly varies. In order to balance instance sizes, a fixed number of consecutive terms is selected for each user, and the remaining terms are discarded.

The high dimensionality of text datasets badly affects the applicability of classification algorithms. Feature selection (158) is a widely used preprocessing stage for reducing the dimensionality of the datasets. In the feature selection module, we employ the χ^2 (CHI square) statistic for every term in order to calculate their discriminative power. Most discriminative features are selected according to the χ^2 scores and used as the feature set. The remaining less discriminative features are eliminated in the feature selection module.

The operation of the modules of the preprocessing stage shows variations in case of message-specific attributes or the style-based feature set. For the case of message-specific attributes, the cleansing/filtering module also employs word blocking. This is because chat messages typically contain only a few words, and it is difficult to correctly classify a message with this little information. The cleansing/filtering module

concatenates multiple consecutive messages of the same user into a single long message. After blocking, the message instances become lengthy enough to have sensible information (40).

In the case of the style-based feature set, instead of terms, a number of stylistic features are extracted. Some of these features contain statistics about the punctuation and stopword usage. Thus, for the construction of style-based feature sets, punctuation marks and stopwords are not eliminated in the cleansing/filtering module. Additionally, for user-specific attributes, the feature sets of all chat messages belonging to a user are combined and used as the feature set for that user. Since the instances contain roughly equal number of features in style-based feature sets, the undersampling module does not try to balance the instance sizes.

The classification stage contains three modules. In the cross validation module, the instances in the dataset are shuffled and divided into 10 equal-sized instance blocks. One of these blocks is selected as the test instance block while other instance blocks are used for the training the framework. The training module uses the training instances supplied by the cross validation module. The output of the training module is a classification model, which is used by the testing module in order to predict the classes of each test instance. The testing module produces a set of predictions based on the classification model and the accuracy of a test is defined as the number of correct predictions divided by the number of total predictions. This operation is repeated 10

times, each time with a different block selected as the test instance block. The average of all predictions gives the prediction accuracy of a classifier. The testing module uses a set of algorithms selected from the Harbinger machine learning toolkit (30) and SVM-light (70). An overview of the selected algorithms can be found in the corresponding references.[1]

3.6 Experimental Results

3.6.1 Experimental Setup

In order to examine the predictability of user and message attributes, the personal information within the chat server logs are used. The attributes retrieved from the server logs such as the users' birth years, and educational environments are submitted voluntarily, they may be missing. As a consequence, some attribute classes are very lightly populated and the use of such classes in evaluating the predictability of that attribute may be impractical. Thus, the experiments are conducted on a selection of the most populated classes of each attribute.

As an illustrative example, the connectivity domain attribute has seven possible class values. For examining the predictability of the connectivity domain attribute, the

[1]The source codes of these algorithms are publicly available online and may be obtained from the following Web addresses:
http://bmi.osu.edu/~barla/coding/HMLT/download/HMLT.tar.gz
http://download.joachims.org/svm_struct/current/svm_struct.tar.gz

most populated two and three classes are selected from the possible seven classes, and the experiments are conducted only on the instances belonging to those classes.

Table 3.5 summarizes the experiments conducted for estimating the prediction accuracies of each attribute. The table contains information about the number of classes, the number of instances, and a set of sample classes used in each test set. Test sets are tagged by concatenating the attribute name, the number of classes, and the number of instances used to represent each class. For example, the School-3-80 tag corresponds to the experiment conducted for predicting the educational environment of users. This experiment involves three possible classes, each of which contains 80 representative instances. As an example for the case of message-specific attributes, the experiment tagged with Author-10-26 involves 10 possible classes, each of which contains 26 instances. Here, each class represent a different author, and instances correspond to message blocks generated by concatenating a particular author's messages.

A selection of classifiers from the Harbinger machine learning toolkit (30) is used for predicting the user and message attributes. The selected classifiers are *k*-NN (58), NB (103), and PRIM. Additionally, SVM-light (70) software is used in order to apply SVM to the chat mining problem. In each test setting, 90% of the most discriminative features are used as the representatives. A sequence of 3000 words is used as the maximum document size for term-based feature sets. The remaining terms in the documents containing more than 3000 terms are discarded. For the *k*-NN classifier, the

Table 3.5: Test sets, their parameters, and sample classes

Test set	# of classes	# of instances in each class	Sample classes
Author-2-35	2	35	Andromeda and Taru
Author-10-26	10	26	Andromeda, Taru, Zizer, ...
Author-100-10	100	10	Andromeda, Taru, Zizer, ...
BirthYear-2-30	2	30	birth year before 1976 (inclusive), birth year after 1976 (exclusive)
BirthYear-4-30	4	30	1975, 1976, 1977, 1978
DayPeriod-2-34	2	34	Day, night (representing 12-hour periods)
DayPeriod-4-17	4	17	Morning, afternoon,evening, night (representing 6-hour periods)
Domain-2-35	2	35	.edu, .com
Domain-2-50	2	50	.edu, .com
Domain-2-65	2	65	.edu, .com
Domain-3-30	3	30	.edu, .com, .net
Gender-2-50	2	50	Male, Female
Gender-2-100	2	100	Male, Female
Gender-2-200	2	200	Male, Female
Receiver-2-35	2	35	Andromeda, Taru
Receiver-10-26	10	26	Andromeda, Taru, Zizer, ...
School-2-190	2	190	Bilkent, METU
School-3-80	3	80	Bilkent, METU, Ege
School-3-120	3	120	Bilkent, METU, Ege
School-5-50	5	50	Bilkent, METU, Ege, KHO, ...
School-10-29	10	29	Bilkent, METU, Ege, KHO, ...

cosine similarity measure is used as the distance metric and the number of the nearest neighbors, *k*, is selected as 10. A polynomial kernel (70) is used in SVM. Each experiment is repeated 5 times and the average prediction accuracies are reported.

3.6.2 Analysis of Predictability

In order to visualize the predictability of different attributes, PCA is used. By using PCA, it is possible to reduce the dimensionality of the instances, allowing them to be plotted in two dimensions (21). Figure 3.3 shows PCA results for four different attributes using a term-based feature set. These attributes are the gender, identity, and Internet connectivity domain of an author and the time period of the messages. As the PCAs of the style-based feature set is similar, they are omitted from this study. Also, note that the coordinate values of the principle component analysis are not displayed. In this work, PCA is only used for the reduction of dimensionality of the dataset. Thus, the values of the data points are not indicative of anything, and only the relative proximities of the data points are important.

Since the data points for each author cover separate regions, it is reasonable to expect high accuracies in predicting the identity of the author of a message. For the PCA of the Internet connection domain, it can be seen that the distribution of data points that belong to the ".com" and ".net" domains cover nearly identical regions while the data points belonging to the ".edu" domain cover a separate region. Hence, it would be reasonable to expect that the ".edu" class could be predicted accurately while "com" and "net" domains would be frequently mispredicted. The results of PCA show that it would not be possible to discriminate all attributes equally using a term-based feature set.

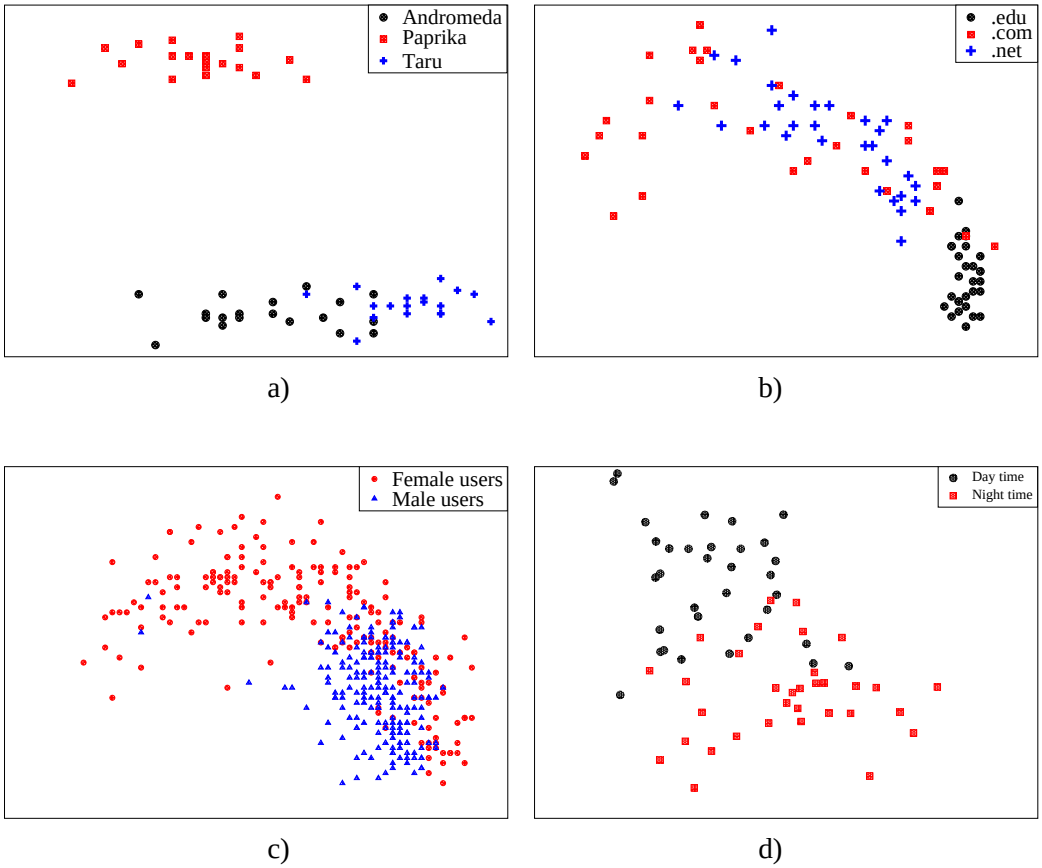


Figure 3.3: The results of the PCA for four different attributes (following our earlier convention): a) Author-3-20, b) Domain-3-20, c) Gender-2-200, and d) DayPeriod-2-34.

3.6.3 User-Specific Attributes

Table 3.6 summarizes the prediction accuracies of the experiments conducted on the user-specific attributes. Among all experiments, the highest prediction rates are achieved for the Internet connection domain of a user. For this attribute, the NB classifier predicts 91.8% and 68.7% of the test instances correctly for the Domain-2-50 and Domain-3-30 test cases respectively. The gender, education environment, and the birth year attributes of a user are also predicted accurately. The prediction accuracies

Table 3.6: Prediction accuracies of experiments conducted on user-specific attributes

	Term-based feature set				Style-based feature set			
Tag	*k*-NN	NB	PRIM	SVM	*k*-NN	NB	PRIM	SVM
BirthYear-2-30	50.1	**60.8**	53.8	56.3	50.0	**75.4**	55.5	48.0
BirthYear-4-30	24.0	**27.3**	20.0	26.5	22.8	**37.4**	19.9	22.0
Domain-2-35	59.7	**90.0**	77.2	64.3	63.9	**90.0**	66.9	59.7
Domain-2-50	58.2	**91.8**	74.0	63.6	64.2	**88.2**	74.4	69.0
Domain-2-65	55.9	**91.4**	79.3	65.2	68.6	**89.8**	78.0	74.1
Domain-3-30	34.0	**67.4**	49.6	39.6	34.7	**68.7**	48.2	45.8
Gender-2-50	73.4	80.0	53.4	**81.5**	63.2	**71.8**	51.2	71.4
Gender-2-100	74.5	81.5	58.3	**82.2**	61.7	**81.9**	64.2	72.3
Gender-2-200	72.2	78.2	56.4	**80.2**	62.4	**81.7**	64.9	77.8
School-2-190	56.8	**68.8**	55.8	66.8	59.3	55.2	50.3	**62.9**
School-3-80	43.6	56.7	35.9	**59.7**	43.1	**47.0**	34.0	51.0
School-3-120	42.7	53.2	41.1	**61.0**	44.1	40.4	32.0	**63.7**
School-5-50	30.8	48.9	26.8	**53.4**	29.1	41.2	25.9	**43.7**
School-10-29	22.5	37.8	17.6	**39.0**	20.4	**26.7**	13.9	26.2

of 82.2% and 75.4% are achieved in prediction of the gender and the birth year of a user respectively. The educational environment of a user attains 68.8%, 53.4%, and 39.0% correct prediction rates for the School-2-190, School-5-50, and School-10-29 test cases respectively. The results of the classification experiments using a term-based feature set lead to the conclusion that gender, identity, and Internet connection domain attributes contain information that reflect the language preferences of a user and it is possible to predict these attributes.

In order to verify whether the experiments are more than some lucky guessing, the level of significance for each experiment is determined. For this purpose, two prediction functions are generated. These functions are used to represent a control group and a treatment group. The control group consists of random guesses for each

Table 3.7: Significance analysis conducted on user-specific attributes

	Term-based feature set		Style-based feature set	
Tag	z-score	p-value	z-score	p-value
BirthYear-2-30	1.73	8.3e-1	2.10	2.7e-2
BirthYear-4-30	1.66	1.9e-1	2.03	5.8e-2
Domain-2-35	4.45	6.2e-7	3.74	8.0e-4
Domain-2-50	5.32	9.5e-9	4.27	3.3e-5
Domain-2-65	5.44	8.4e-8	5.61	7.1e-9
Domain-3-30	4.11	3.6e-6	3.94	6.6e-5
Gender-2-50	4.02	3.3e-5	2.32	3.7e-2
Gender-2-100	5.31	3.4e-7	5.39	5.1e-8
Gender-2-200	6.51	6.4e-10	7.11	1.1e-11
School-2-190	3.74	2.0e-4	2.92	3.2e-3
School-3-80	4.72	9.8e-7	2.60	1.2e-3
School-3-120	6.78	1.3e-12	6.64	5.2e-12
School-5-50	7.17	1.1e-12	4.49	2.1e-5
School-10-29	7.10	4.1e-10	3.44	2.1e-5

instance while the treatment group consists of predictions after the classifiers are used. The value of the prediction function is 1 if the instance is correctly predicted and 0 otherwise. Wilcoxon signed-rank test (153) is used for determining the levels of significance. The significance levels are computed for the best classification result, represented in bold case in Table 3.6. Table 3.7 summarizes the z-scores and p-values for each experiment group for user-specific attributes. Noting that the most common level of significance is 5%, all experiments performed significantly better than random guesses. The experiments conducted on the Internet connectivity domain, gender, and educational environment attributes all result in very low levels of significance, which means that the methods proposed in this work can be used effectively to predict these attributes in chat messages.

In predicting the user-specific attributes, the use of term- and style-based feature sets perform almost equally well. While the term-based feature sets performs better than style-based feature sets for predicting the Internet connection domain and the educational environment of a user, the use of style-based feature sets perform better for predicting the birth year of a user.

The performance of different classifiers vary throughout the experiments. The experimental results on the prediction of user-specific attributes show that NB and SVM perform best in all settings although the results show that no single classifier can be the "best performer." While NB performs better than SVM in predicting the connection domain of a user, SVM performs slightly better in predicting the educational environment of a user. *k*-NN produces the worst results for the prediction of the Internet connection domain while PRIM performs the poorest in prediction of all other attributes. PRIM's poor performance is a result of it being a rule-based classifier. PRIM generates a set of classification rules covering all the instances in a class, and use these rules to classify the test instances. Due to the high dimensionality of the dataset, these rules contain only the most discriminative features, and thus, tend to be valid for only a small subset of the instances in a class. Since such rules fail to classify a large enough subset of the test instances, the classification of PRIM degenerates into random guesses.

Table 3.8: Prediction accuracies of experiments conducted on message-specific attributes

	Term-based feature set				Style-based feature set			
Tag	*k*-NN	NB	PRIM	SVM	*k*-NN	NB	PRIM	SVM
Author-2-35	**100.0**	**100.0**	98.7	**100.0**	98.3	**99.7**	92.9	97.1
Author-10-26	98.7	**100.0**	74.4	99.9	84.0	89.1	51.7	**97.1**
Author-100-10	88.3	89.9	44.0	**99.7**	31.2	29.7	5.8	**78.9**
DayPeriod-2-34	66.2	**71.6**	48.8	60.7	59.9	**63.8**	54.3	59.6
DayPeriod-4-17	34.6	**47.6**	25.4	39.6	30.7	38.9	28.5	**41.6**
Receiver-2-35	60.0	**75.0**	51.6	67.0	58.5	**60.5**	53.7	53.4
Receiver-10-26	25.1	40.9	21.8	**41.1**	**12.4**	11.2	9.2	10.6

3.6.4 Message-Specific Attributes

Table 3.8 summarizes the prediction accuracies of experiments conducted on the message-specific attributes. The identity of the author is predicted with perfect accuracy for two and 10 authors using term-based feature sets. The prediction accuracy drops to 99.7% even when the number of users is increased to 100. The experiments for predicting the identity of the author of a message show that each author has a distinct communication style and word selection habits. The use of style-based feature sets also show that the receiver of a message and the time period the message is written is also predictable. The receiver of a message is predicted with 75.0% and 40.9% accuracy for the Receiver-2-25 and Receiver-10-26 test cases, respectively. The classification accuracies for the DayPeriod-2-34 and DayPeriod-4-37 test cases are 71.6% and 47.6%, respectively. Table 3.9 also summarizes the significance tests conducted on message-specific attributes.

The use of style-based feature sets perform equally with term-based feature sets

Table 3.9: Significance analysis conducted on message-specific attributes

	Term-based feature set		Style-based feature set	
Tag	z-score	p-value	z-score	p-value
Author-2-35	5.24	1.5e-7	4.79	1.2e-5
Author-10-26	13.37	7.1e-73	13.12	9.2e-69
Author-100-10	27.19	3.3e-318	24.16	2.5e-200
DayPeriod-2-34	3.30	1.9e-3	1.46	1.8e-1
DayPeriod-4-17	2.32	3.9e-3	1.80	7.6e-2
Receiver-2-35	2.39	2.0e-3	1.20	2.4e-1
Receiver-10-26	6.18	5.2e-9	1.42	3.6e-1

when the number of classes is small. However, as the number of classes increases, the decrease in the prediction accuracy is more significant when using style-based feature sets than using term-based feature sets. The reason of this rapid decrease in the prediction accuracies is that the dimensionality of the style-based feature sets are much smaller than that of the term-based feature sets; and as the number of classes increases, all classifiers exhibit difficulties in differentiating the instances of different categories.

Contrary to the results of the experiments employed using the term-based feature sets, the receiver and day period of a message can only be predicted almost with random accuracy using a style-based feature set. This interesting finding shows that the vocabulary use of a person is dependent on the target and the time of the message while the communication style is only dependent on the person writing that message.

For predicting the message-specific attributes, NB and SVM achieve best results among all classifiers. While both classifiers perform similarly for small number of

classes, the experiments on the authors' identity show that as the number of classes increases SVM performs better than NB. The PRIM classifier performs the worst for all attributes for both term- and style based feature sets.

3.7 Concluding Remarks

In this chapter, the predictability of various user- and message-specific attributes in electronic discourse is examined. Specifically, the word selection and message organization of chat users are investigated by conducting experiments over a large real-life chat dataset. Our observations show that many characteristics of chat users and messages can be predicted using their word selection and writing habits. The experiments point out that some attributes have recognizable traces on the linguistic preferences of an author. A possible alternative view to the chat mining problem is to examine how the linguistic traits of a person effect the writing style. In this section, we take this alternative view and discuss how a person's attributes affect his writing style.

Table 3.10 shows the set of most discriminative terms for different attributes. As chat conversations occur in a spontaneous environment, the use of slang words and mispellings is frequent. Two different users may write the same word quite differently. For example, the word "something" (spelled as "birsey" in Turkish with ASCII characters) is used in its syntactically correct form by the user "Andromeda" while "Paprika" uses a slang version ("bishiy" in Turkish with ASCII characters) of the same word in

Table 3.10: The most discriminating words for each attribute. The discriminative power of each word is calculated using the χ^2 statistic

Attribute name	Example Class	The most discriminating words
Author	Andromeda	byes (bye – slang), ok, birsey (something)
	Paprika	diil (nothing – misspelled), ehe (hah – slang)
		bishiy (something – misspelled)
	Taru	hmm (emoticon), dakika (minute), ha (hah!)
BirthYear	1979	dusunuyon (thinking – misspelled), ucuza (cheaply
		acar (opens)
	1978	onemli (important), demek (then), git (go)
DayPeriod	Afternoon	kusura (fault), uzgunum (I'm sorry), lutfen (please)
	Evening	geceler (nights), hosca (finely), grad (graduate)
Domain	.edu	git (go), gelir (comes – 2nd person), saat (clock)
	.com	cikardin (you displace – 2nd person), muhabbet (chat)
		karsindaki (opposite)
Gender	male	abi (brother), olm (buddy – misspelled)
		lazim (required)
	female	ayyy (ohhh!), kocam (my husband)
		sevgilimin (my lover's)
Receiver	Celefin	olm (buddy – misspelled), falan (so)
		yaw (hey! – misspelled)
	Kebikec	hmm (a notification), seker (sugar), adam (man)
School	Ege Univ.	Ege (a region), Bornova (a city in Agea region)
		Izmirde (in Izmir, a city in Agea region)
	Bilkent Univ.	Bilkent (Univ. Name), BCC (Bilkent Computer Center)
		Bilkentte (in Bilkent)
	METU Univ.	ODTU (univ. Name in Turkish), METU (Univ. name)
		yurtlar (dormitories)

his messages. The receiver of a message also affects the word selection habits. Some users tend to receive messages containing more slang words than others. The vocabulary use is additionally affected from the period of the day. Our observations show that during the day hours, users tend to converse more politely, using apologetic words more frequently.

The user-specific attributes also affect the word selection habits. The most discriminative words of the users connected from the ".edu" domain contain more inquiries and imperatives. On the contrary, the users connected from the ".com" domain employ mostly responses and second person references. The users of the ".com" domain tend to use shorter words than the users connected from other domains in their conversations. Another attribute that clearly affects the vocabulary of a user is gender. It is apparent that males tend to use more decisive, dominating sentences using words that can be considered as slang while female conversations involve more content-dependent words and emoticons (e.g., Ayyy!). These findings show similarities with the findings presented in (160). The most discriminative words for the classes of user's educational environment are mostly dominated by the regional terms. In Table 3.10, the most discriminating words of users from three universities in different regions are given. The vocabulary of the users contain many location-specific terms and is clearly affected by the location of the university and its facilities.

The stylistic analysis also provides interesting results. Each chat user expresses himself/herself using an almost-unique and identifiable set of linguistic preferences. The messages of three different users is examined in order to present their stylistic differences. The user named "Andromeda" employs smileys and average-length words more than others, while "Paprika" tend to converse using shorter messages, prevent using punctuation marks, smileys, and function words. The user "Taru" communicates with longer messages containing a large number of punctuation marks and function

words. The time of a message also affects the style and vocabulary of a message. During the day hours, messages are generally shorter and contain less auxiliary elements such as smileys and punctuation marks, while during the night hours the messages tend to be longer containing many function words and punctuation marks.

The writing style shows variations between different domains. The users connected from the ".edu" domain have a smaller vocabulary and use punctuation marks and numerals frequently. On the contrary, the users of the ".com" domain have a larger vocabulary, use a small number of numerals, and write longer messages. The educational environment of a user is another factor that affects the writing style. The users from different universities prefer to use separate sets of smileys. The style of a person is also affected by his/her gender. In general, female users prefer longer and content bearing words. They also prefer shorter sentences than male users and omit the use of stopwords and punctuation marks. Long messages and use of short words are most discriminating stylistic characteristics of male users. The use of style-based feature sets prove to be more effective than the use of term-based feature sets for determining the birth year of an author. This result also shows that the age group of an individual is an important factor that affects the stylistic characteristics of a person's messages. The experiments conducted for determining the birth year attribute of a user show that younger users mostly have a smaller vocabulary. Additionally, as (119) also pointed out, younger users prefer using smileys more than older users.

Chapter 4

A Parallel Framework for In-Memory Construction of Term-Partitioned Inverted Indexes

4.1 Introduction

The evolution of communication technologies in recent years gave rise to a rapid increase in the amount of textual digital information and the demand to search over this type of information. One of the largest industries of our era, the searching industry, has flourished around these demands.

Inverted indexes, due to their superior performance in answering phrase queries (128), are the most commonly used data structures in Web search systems. An inverted index consists of two parts: a *vocabulary* and *inverted lists*. The vocabulary contains the collection of distinct *terms*, which are composed of character strings (*words*) that occur in the documents of the collection. For each term in the vocabulary, there is an associated *inverted list*, or *posting list*. The inverted list for a term is a list of *postings*, where a posting contains an identifier for a document that contains that term. Depending on the granularity of information, the frequency and the exact term positions may also be stored in the postings.

Inverted index data structure is quite simple, yet Web-scale generation of a global inverted index is very costly due to the size, distributed nature and growth/change rate of the Web data (35). Fast and efficient index construction schemes are required to provide fresh and up-to-date information to users. Furthermore, since the data to be indexed is crawled and stored by distributed or parallel systems (due to performance and scalability reasons), parallel index construction techniques are essential.

There are two major partitioning schemes used in distributing the inverted index on parallel systems: document-based and term-based partitioning. In document-based partitioning, the documents are assigned to index servers and all the postings related with the assigned documents are stored in a particular index server. In term-based partitioning, each term in the vocabulary and the related inverted lists are assigned to an index server.

Almost all of the major search engines use document-based partitioning due to the ease in parallel index construction of document-based partitioned inverted indexes. Term-based partitioning on the other hand has advantages that can be exploited for better query processing (108). In this study, we propose an efficient parallel index construction framework that can be used for generating term-based partitioned inverted indexes starting from a document-based partitioned collection most possibly generated via a parallel crawling of Web documents.

4.1.1 Related Work

Early studies on index construction are focused around disk-based algorithms designed for sequential systems (59; 106; 154). In (154), authors present a method that traverses the disk-based document collection twice; once for generating a term-based partition to divide the work into loads, and once for inverting the dataset iteratively for each pass defined in the previous pass. The emphasis is on using as little memory as possible. In (106), authors use a multi-way, in-place, external merge algorithm for inverted index construction with less primary memory. In (59), authors propose an in-memory index construction method for disk-based inverted indexes where the document set is divided into batches that are inverted in memory and then merged and written into disk. In their work, authors facilitate the use of compression in order to achieve a more effective inversion.

More recent works on sequential systems are mainly focused on on-line incremental updates over disk-based inverted indexes (29; 91). In (29), the authors propose a hybrid indexing technique. The proposed method merges small posting lists with the already existing index, while using posting list re-allocation for large posting lists. The authors also propose two in-place merge techniques for updating long posting lists. In (91), the authors evaluate two index maintenance strategies and propose alternatives for improving these strategies. These improvements are based on over-allocation of posting lists and keeping incremental updates within vocabulary before index re-merging.

The following studies on index construction (48; 69; 104; 108; 120; 121; 135), extend disk-based techniques for parallel systems. In (48), a document-based allocation scheme for inverted indexes is presented. The authors emphasize both storage balance and inter-processor communication times and try to minimize both using genetic algorithms. In (69), the authors evaluate the effects of term- and document-based partitioning methods on a shared-everything architecture. They use query statistics to balance the required I/O times among processors on a disk-based architecture.

In (120) and (121), the authors present a disk-based parallel index construction algorithm, where initially the local document collections are inverted by all processors in parallel. The processors generate a global vocabulary on a host processor and the host processor divides the document collection among all processors in lexicographic order assuming global knowledge over the document collection. The authors also analyze

the merging phase of the inverted lists in (121), presenting three algorithms. In their work, the authors mainly focus on the parallel generation of the distributed index and the communication costs are not taken into consideration.

In (135), the author describes an index inversion framework for distributed information retrieval systems. Although the method presented in (135) achieves storage balance among processors, it does not consider minimizing the communication loads of the processors. In (135), it is also assumed that it is possible for the inverted indexes to be incrementally updated over time, and specialized data structures for minimizing the index update times are proposed. The cost of the inversion process is also emphasized, and four different index inversion methods are presented. In (104), the authors again start from a document partitioned collection and use a software-pipelined architecture to invert document collections. The collection is divided into runs, and for each run, documents are parsed, inverted, sorted, and flushed into disk in a pipelined fashion. In (108), the authors propose a load balancing strategy in a term-partitioned inverted index on a pipelined query processing architecture (107). In (108), both replication of inverted lists and a query statistics-based assignment scheme is presented, yielding up to %30 net query throughput improvements.

4.1.2 Motivation and Contributions

We would like to repeat a catchy phrase often credited to Jim Gray: “Memory is the new disk, disk is the new tape”. With the advent of 64-bit architectures, huge memory spaces are available to single machines and even very large inverted indexes can fit into the total distributed memory of a cluster of such systems, enabling memory-based index construction. Furthermore, cloud computing systems such as Amazon EC2 are commercially available today. They offer leasing of virtual machines without owning and maintenance costs and thus ease the utilization and management of large cluster of servers. Thus, we believe that the benefits of parallel index construction is not limited to dividing and distributing the computational task to different processors. The current advances in network technologies, cloud computing and the high availability of low cost memory provides an excellent medium for memory resident solutions for parallel index construction.

In this work, we extend our previously proposed in-memory parallel inverted index construction scheme (83) and compare the effects of different communication-memory organization schemes to the parallel inversion time. In our framework, we propose to avoid the communication costs associated with global vocabulary construction with a term-to-bucket assignment schema. This schema prevents term information to be sent to a host, where a reasonable term-to-processor assignment would be computed using the term distribution among processors, thus avoiding a possible bottleneck of communication. Furthermore, term-to-bucket partitioning allows the framework to completely

avoid creating a global vocabulary, eliminating the need of a further communication phase.

We also investigate several assignment heuristics for improving the final storage balance, the final query processing loads, and the communication costs of inverted index construction. Here, storage balance is important since we are trying to build a memory-based inverted index. Query processing load balance is important since the reason for building the inverted index is for faster query processing and this can be done better if the loads of the processors are balanced. Finally, the communication cost is important since it effects the running time of parallel inversion.

Furthermore, we investigate the effects of various communication-memory organization schemes. Since parallel inversion is a communication-bound process, we observe that the utilization of the communication-memory and the network has significant effects on the overall inversion time. Our findings indicate that, dividing the communication memory into $2 \times K$ buffers, where K of which are used for sending messages and the remaining K are used for receiving messages, yields the best performance. This is due to the fact that this communication-memory organization scheme maximizes the communication/computation overlap.

Finally, we test the performance of the proposed schemes by performing both simulations and actual parallel inversion of a realistic Web dataset and report our observations. Our contributions in this work are prior to optimizations such as compression (165). However, it is possible to apply data compression to the proposed model, making it possible to work with even larger data collections.

The organization of this chapter is as follows: In Section 4.2, we introduce the memory-resident distributed index inversion problem and describe our framework. In Section 4.3, we provide our overall parallel inversion scheme. In Section 4.4, we describe the investigated assignment schemes in detail. In Section 4.5, we present several memory organization schemes in order to reduce the communication time and discuss their advantages and disadvantages. We provide our experiments, their analysis, and extensive discussions on the results of our experiments in Section 4.6. Finally, in Section 4.7 we conclude and discuss some future work.

4.2 Framework

Most of the largest text document collections that are actively in use today are Web-based. These repositories are mainly created and used by Web search engines. An important consideration in the design of parallel index construction systems should be their applicability to such real life data collections. In this work, our efforts are based on presenting an efficient and scalable index construction framework specifically

designed for Web-based document collections.

Parallel search engines collect Web pages to be indexed via distributed Web Crawlers (26). In general, at the end of a crawling session, a document-based partition of the whole document collection is obtained, where each part is stored in a physically separate repository (26). The state-of-the-art approach to distributing the crawling and storage tasks uses a site-hash-based assignment, where the site names of pages are hashed and documents are assigned to repositories according to those hash values (18; 36; 34).

The framework presented in this study has three assumptions on the initial data distribution. First, the initial document collection is assumed to be distributed among the processors of a parallel system. That is, each processor is assumed to have a portion of the crawled Web documents and maintain information about only its own local dataset. Thus in this work, no processor contains a global view of the document collection. Second, each processor is assumed to contain a disjoint set of documents. This means that the overall system contains no replica of any document. Third, the Web pages are assumed to be distributed among these processors using a site-based hashing. That is, all pages from a site are assigned to a single processor, hence each site is assumed to be an atomic storage task. Consequently, the initial storage loads of the processors are not necessarily perfectly balanced. These three assumptions are in concordance with the output format of general purpose crawling systems.

In this framework, the objective of parallel index construction is to generate a final term-partitioned parallel inverted index from a document-partitioned collection stored on a distributed shared-nothing architecture. The final term-partitioned inverted index will also be stored in a distributed fashion in order to allow both inter- and intra-query parallelism on query processing. In this context, our approach has similarities with parallel matrix transpose operations.

4.3 Parallel Inversion

Our inversion scheme starts with a document-based initial partition. Such an initial document-based partition is depicted in Figure 4.1(a). Our overall parallel inversion scheme has the following phases:

- **Local inverted index construction:** Each processor generates a local inverted index from its local document collection. This process is illustrated in Figure 4.1(b). Note that inverted lists for some terms can appear in multiple processors.

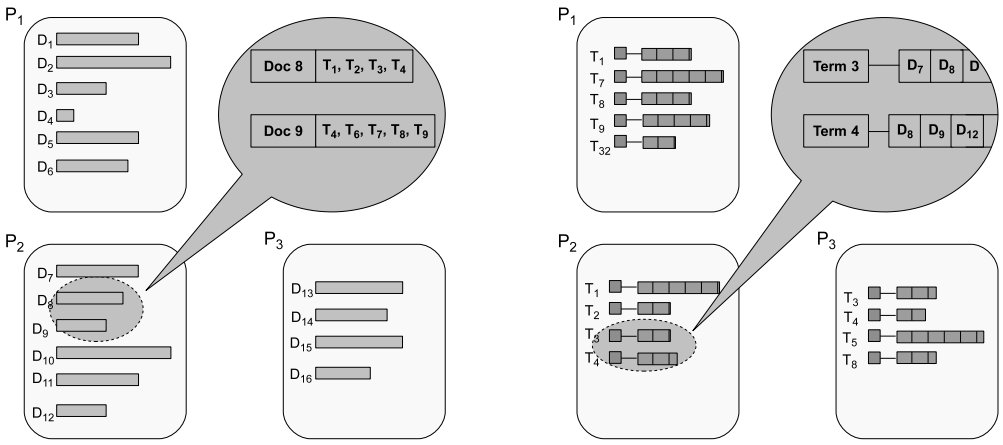
- **TermBucket-to-processor assignment:** Each processor uses hashing to find a deterministic assignment of terms into a pre-determined number buckets. Buckets are used to randomly group inverted lists so that the communication costs in

the termBucket-to-processor assignment phase is reduced. All processors communicate the sizes of their term buckets to the host processor. The host processor generates a termBucket-to-processor mapping under the constraint that in the final assignment, the storage and query processing load balance is achieved and communication cost is minimized. This process is illustrated in Figure 4.1(c). Note that many buckets exist in multiple processors due to the initial document partitioning.

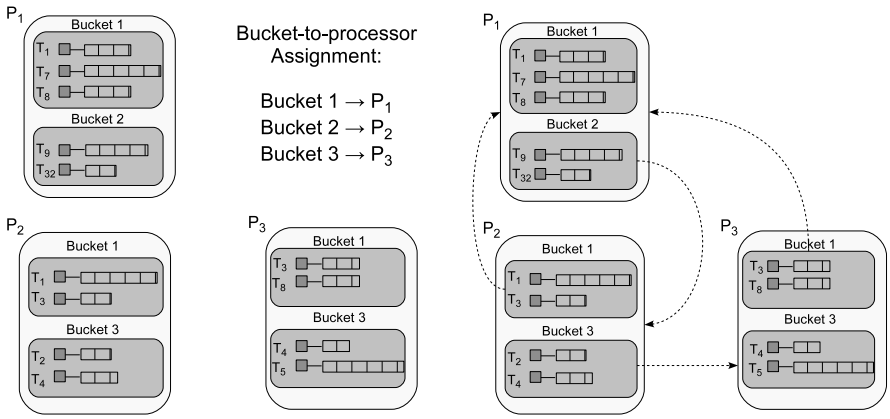
- **Inverted list exchange-and-merge:** The processors communicate appropriate parts of their local inverted indexes in an all-to-all fashion. This process is illustrated in Figure 4.1(d). The remaining local inverted index portions are merged with the received portions and final inverted index is generated. The final term-partitioned inverted index of the initial document-partitioned index in Figure 4.1(b) can be seen in Figure 4.1(e).

4.3.1 Local Inverted Index Construction

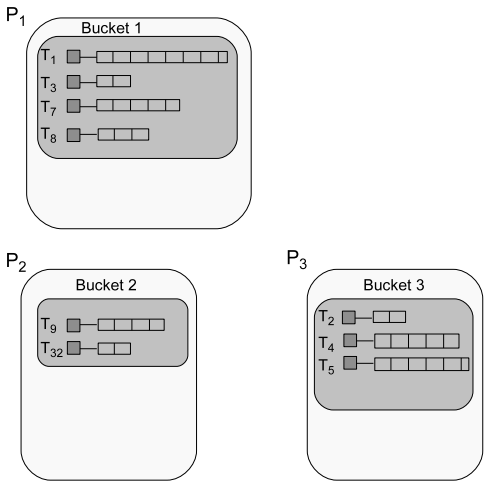
In the local inverted index construction phase, each processor generates a local vocabulary and local inverted lists from its local document collection. Since each processor only contain a unique subset of documents, this operation can be achieved concurrently without any communication. In this phase, the local vocabularies and inverted list sizes are determined and each term is given a unique identifier. In our local index



(a) Initial document partition. (b) Local inverted index construction.



(c) TermBucket-to-processor assignment. (d) Inverted list exchange-and-merge.



(e) Final term-partitioned index

Figure 4.1: Phases of the index inversion process.

construction scheme, the local document collection is read twice. In the first pass, the local vocabularies and inverted list sizes are determined and each term is given a unique identifier. The memory required for local inverted lists is allocated according to the determined inverted list sizes. In the second pass, the document collection is parsed and stored in the respective inverted lists.

4.3.2 TermBucket-to-Processor Assignment

After the local inversion phase, processors contain a document-based partitioned inverted index. In this partition, processors contain different portions of inverted lists for each term. In order to create a term-based partitioned inverted index, each inverted list, in its full form, should be accumulated in one of the processors. To this end, each term in the global vocabulary should be assigned to a particular processor.

This term-to-processor assignment depicts an inverted index partitioning problem. A suitable index partitioning can be defined by many different criteria. In this work, we set the following quality metrics for a "good" term-to-processor assignment:

`QM1:` Balancing the "expected" query processing loads of processors.

`QM2:` Balancing the storage loads of processors.

`QM3:` Reducing the communication overhead during the inversion process through minimizing:

(a) Total communication volume.

(b) Communication load of the maximally loaded processor.

The final query processing loads of processors indicate the amount of processing that a processor is expected to perform once the inversion is finished and the query processing begins. We can estimate this load utilizing previous query logs.

The storage balance of processors guarantees an even distribution of the final inverted index allowing larger indexes to fit in the same set of processors.

Since inversion is a communication-bound process, the minimization of the communication overhead ensures that the inverted list exchange phase of the parallel inversion process takes less time. In this work, minimization of the communication overhead is modeled as the minimization of total communication volume while maintaining the balance on the communication loads of the processors. These are the two commonly used quality metrics that determine the communication performance of a task-to-processor assignment when the message latency overhead remains negligible compared to the message volume overhead (147), (23), which is the case for parallel index inversion.

To optimize the above-mentioned metrics, we investigate existing assignment schemes, comment on possible enhancements over these schemes, and propose a novel assignment scheme which performs better than its' counterparts. Our discussions about bucket-to-processor assignment schemes are explained in detail in Section 4.4.

For the purpose of finding a suitable term-to-processor assignment, the previous works in the literature either assume the existence of a global vocabulary or generate a global vocabulary from the local vocabularies. The global vocabulary can be created by sending each term string, in its word form, to a host processor, where they are assigned global term-ids, and these global term-ids are broadcasted to all processors. However in such a scheme, a particular term would be sent to the host machine by all processors if all processors contain that specific term. Our observations indicate that the cost of such an expensive communication stage is proportional to the cost of inverted list exchange phase. Furthermore, since the host processor receives all the communication, it constitutes a serious bottleneck.

In this work, we propose a novel and intelligent scheme that enables us to avoid global vocabulary construction cost. We propose to group terms into buckets prior to the term-to-processor assignment. Using string hashing functions, each word in a local vocabulary is assigned to a bucket. Afterwards, only the bucket size information is sent to the host processor. The host processor computes a termBucket-to-processor assignment, which induces a term-to-processor assignment, and broadcasts this information to the processors. The effect of bucket processing order on the quality of the assignment is not investigated in this work and the same random bucket processing order is used in evaluating the assignment schemes. We should also note here that it is not necessary to build a global index at the host processor ever. It suffices for the host processor to store only a bucket-to-processor assignment array. Whenever the host

processor receives a query term, all it has to do is to compute the hash of the term, find the bucket for that term and forward the term to the owner processor of the bucket.

4.3.3 Inverted List Exchange-and-Merge

At the end of the termBucket-to-processor assignment phase, all bucket-to-processor assignments are broadcast to the processors by host processor, so that each processor is aware of the bucket-to-processor assignments. In order to create a term-partitioned inverted index, the document-based partitioned local inverted list portions should be communicated between processors in such a way that the whole posting list of each term resides in one of the processors. To this end, all processors should exchange their inverted lists portions in an all-to-all fashion. However, utilizing termBuckets instead of terms for assignment dictates a major change (and an additional cost) in the inverted list exchange-and-merge phase.

Since termBucket-to-Processor assignment prevents the need of creating a global vocabulary, when a processor receives a posting list portion of a term from another processor, it also requires additional information to identify the posting list it receives. To this end, upon sending the posting list portions, the processors should also send the associated term, in its word form, to the receiving processor. Due to this, the all-to-all inverted list exchange communication becomes slightly more costly. However, since the processor-to-host bottleneck due to global vocabulary construction is already

avoided, the performance degradation in all-to-all inverted list exchange communication is more than compensated. Furthermore, this vocabulary exchange is distributed among all processors evenly, further reducing its overhead.

The inverted list exchange between processors is achieved in two steps. First, terms, in their word form, and their posting sizes are communicated. This is done by an all-to-all personalized communication phase, where each processor receives a single message from each other processor. At the end of this step, all processors obtain their final local vocabularies and can reserve space for their final local inverted index structures. Second, inverted list portions are exchanged in bucket id order, and within the buckets in alphabetical order. This step is again performed as an all-to-all personalized communication. However, since this step consumes significant amount of time, the inverted list portions are sent via multiple messages. Memory organization and communication scheme used in this phase is explained in detail in Section 4.5. At the end of inverted list exchange, the remaining inverted lists, and obtained inverted lists for each term are merged and written into their reserved spaces in memory.

4.4 Term-to-Processor Assignment Schemes

In this section, we try to solve the termBucket-to-processor assignment problem with the objectives of minimizing the communication overhead during the inversion and maintaining a balance on the query processing and storage loads of processors after

the inversion. We present adaptations of two previously proposed assignment algorithms (9) to the problem at hand, discuss the shortcomings of these algorithms and propose a novel assignment algorithm that provides superior parallel performance.

In the forthcoming discussions we use the following notations: The vocabulary of terms is indicated with $\mathcal{T}$. Due to the initial site-hash-based crawling assumption, the posting list of each term $t_j \in \mathcal{T}$ is distributed among the K processors. In this distribution, $w_k(t_j)$ denotes the size of the posting list portion of term t_j that resides in processor p_k at the beginning of the inversion, whereas $w_{tot}(t_j) = \sum_{k=1}^{K} w_k(t_j)$ denotes the total posting list size of term t_j.

We assume that prior to bucket-to-processor assignment, each processor has built its local inverted index $\mathcal{I}_k$ and partitioned the vocabulary $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ containing n terms, into a predetermined number m of buckets. The number of buckets m is selected such that $m \ll n$ and $m \gg K$. Let

$$\mathcal{B} = \Pi(\mathcal{T}) = \{\mathcal{T}_1 = b_1, \mathcal{T}_2 = b_2, \ldots, \mathcal{T}_m = b_m\}. \tag{4.1}$$

denote a random term-to-bucket partition, where $\mathcal{T}_i$ denotes the set of terms that are assigned to bucket b_i. In this partition, $w_{tot}(b_i)$ denotes the total size of the posting lists of terms that belong to b_i and $w_k(b_i)$ denotes the total size of the posting list portions of terms that belong to b_i and that reside in processor p_k at the beginning of the inversion.

We also assume that we are given a query set $\mathcal{Q}$ where each query $q \in \mathcal{Q}$ is a subset of $\mathcal{T}$, i.e., $q \subset \mathcal{T}$. The number of queries that a term t_j is requested by is denoted with $f(t_j)$.

In an m-bucket and K-processor system, the bucket-to-processor assignment can be represented via a K-way partition

$$\Pi(\mathcal{B}) = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k\}. \tag{4.2}$$

of the buckets among the processors. The quality of a bucket-to-processor assignment $\Pi(\mathcal{B})$ is measured in terms of three metrics: The query processing load balance (`QM1`), storage load balance (`QM2`) and the communication cost (`QM3`). The query processing load $QP(p_k)$ of a processor p_k induced by the assignment $\Pi(B)$ is defined as follows:

$$QP(p_k) = \sum_{b_i \in \mathcal{B}_k} \sum_{t_j \in b_i} w_{tot}(t_j) \times f(t_j). \tag{4.3}$$

The storage load $S(p_k)$ of a processor p_k induced by the assignment $\Pi(B)$ is defined as follows:

$$S(p_k) = \sum_{b_i \in \mathcal{B}_k} \sum_{t_j \in b_i} w_{tot}(t_j). \tag{4.4}$$

The communication cost of a processor p_k induced by the assignment $\Pi(B)$ has two components. Each processor must receive all portions of the buckets assigned to it from other processors. Thus total reception cost/volume of a processor p_k is:

$$Recv(p_k) = \sum_{b_i \in \mathcal{B}_k} \sum_{t_j \in b_i} (w_{tot}(t_j) - w_k(t_j)). \tag{4.5}$$

Each processor must also send all postings that are not assigned to it to some other processor. The total transmission cost of p_k is represented by $Send(p_k)$ and is defined as:

$$Send(p_k) = \sum_{b_i \notin \mathcal{B}_k} \sum_{t_j \in b_i} w_k(t_j) \tag{4.6}$$

The total communication cost of a processor is defined as:

$$Comm(p_k) = Send(p_k) + Recv(p_k) \tag{4.7}$$

4.4.1 Minimum Communication Assignment (MCA)

MCA algorithm minimizes the total communication volume while ignoring storage and communication balancing (9). The MCA scheme is based on the following simple observation. If a termBucket is assigned to the processor that contains the largest

portion of the inverted lists of the terms belonging to that bucket, the total message volume incurred due to this assignment will be minimized. Thus, if we assign each termBucket $b_i \in \mathcal{B}$ to the processor p_k that has the largest $w_k(b_i)$ value, the total volume of communication for this term will be minimized. By assigning all terms using the above criteria, an assignment with global minimum total communication volume can be achieved.

4.4.2 Balanced-Load Minimum Communication Assignment (BLMCA)

The BLMCA scheme is an effort to incorporate storage balancing to MCA (9). In this scheme, termBuckets are iteratively assigned to processors. In BLMCA, for each termBucket, first the target processor that will incur the minimal total communication is determined using the criteria in MCA scheme. If assignment of the particular termBucket to that processor does not make the storage loads of the processors more skewed (does not increase the maximum storage load of all processors) at that iteration, the assignment proceeds as in MCA scheme. Otherwise, the termBucket is assigned to the minimally loaded processor.

4.4.3 Energy-Based Assignment (EA)

In BLMCA, two separate cost metrics are evaluated: The storage load balance and total communication cost. However, at each iteration, only one of these metrics is chosen

to be optimized. Furthermore, both MCA and BLMCA models the communication cost as the total communication volume and disregards the maximum communication volume of a single processor. In order to minimize the maximum communication cost of a processor, we should consider both the reception cost of the assigned processor and the transmission costs of all other processors.

In the EA scheme, we propose a model that prioritizes reducing the maximum communication cost of processors as well as maintaining storage and query processing load balance. To this end we define the energy E of an assignment $\Pi(B)$. This energy definition is based on the storage loads, query processing loads and communication costs of processors. Recall that $Comm(p_k)$ of a processor incorporates both reception and transmission costs of processor p_k. We define two different energy functions for a given termBucket-to-processor assignment $\Pi(B)$:

$$\begin{aligned} E^1(\Pi(B)) = Max\{&Max_{1\leq k\leq K}\{Comm(p_k)\},\\ &Max_{1\leq k\leq K}\{S(p_k)\},\\ &Max_{1\leq k\leq K}\{QP(p_k)\}\} \end{aligned} \tag{4.8}$$

$$E^2(\Pi(B)) = \sum_{1}^{K}(Comm(p_k))^2 + \sum_{1}^{K}(S(p_k))^2 + \sum_{1}^{K}(QP(p_k))^2 \tag{4.9}$$

Utilizing these two energy functions, we propose a constructive algorithm that assigns termBuckets to processors in a successive fashion. The termBuckets are processed in some order, and the energy increase in the system by K possible assignments of each bucket are considered. The assignment that incurs the minimum energy increase is performed. That is, for the assignment of a termBucket b_i in the given order, we select the assignment that minimizes

$$E(\Pi(B_{i-1} \cup \{b_i\})) - E(\Pi(B_{i-1})). \tag{4.10}$$

where B_{i-1} denotes the set of already assigned termBuckets.

We should note here that proposed energy-based assignment schemes also have the nice property of being easily adaptable for incremental index updates. To enable this feature at the end of inversion process, it is sufficient to store the energy levels of each process. These values then can be used to perform (re)assignment of indexes in an incremental fashion. The minimization of inversion time feature of these schemes would be very helpful in minimizing the incremental update time as well. However, we should note that enabling incremental update in these schemes would necessitate the construction of a global vocabulary on the server node.

We consider both E^1 and E^2 energy definitions and report the results of both schemes in our experiments. We call the E^1-based assignment scheme as E^1A and the E^2-based assignment scheme as E^2A.

4.5 Communication-Memory Organization

In the final stage of the memory-based parallel inverted index construction, the portions of each posting list are communicated between processors to accumulate each posting list in one processor, where they would be merged in order to construct the final inverted index. This phase can be summarized as an all-to-all personalized communication phase with different number of messages and total message sizes. In this phase, each processor should identify local posting list portions to be sent to other processors, prepare message buffers to send them using the available memory for this communication and send them to the target processors. At the same time, each processor should retrieve posting list portions assigned to them from other processors and merge them in order to generate the final posting lists.

Posting list exchange operation requires intensive communication between processors and dominates the total time required to complete the index inversion. An important question when communicating the posting list portions is how to use/organize the available memory so that the communication phase takes the least possible time. In this work, we evaluate four different communication memory organization schemes and their impact on total run time of index inversion. These schemes are:

- 1-Send 1-Receive buffer scheme (1s1r)
- 1-Send (K-1)-Receive buffer scheme (1sKr)
- (K-1)-Send 1-Receive buffer scheme (Ks1r)
- (K-1)-Send (K-1)-Receive buffer scheme (KsKr)

In investigating different communication-memory organization schemes, we assume that the total memory spared for communication is fixed, say M. In 1s1r, the communication memory is split into one send and one receive buffer, each with size $M/2$. In 1sKr and Ks1r, the memory is split into K buffers each with size M/K. In 1sKr, one of these buffers is used as a send buffer and the remaining K-1 buffers are reserved for receiving messages from other processors. In Ks1r, each processor maintains one receive buffer and K-1 send buffers, which are reserved for sending messages to other processors. In KsKr, the memory is split into $(2\times K)-2$ buffers each with size $M/((2\times K)-2)$. K-1 of these buffers are reserved as send buffers as in Ks1r, while the other K-1 buffers are reserved as receive buffers as in 1sKr.

In all of these schemes, the communication commences through several stages. First, all processors issue non-blocking receives for each receive buffer. Then, each processor starts preparing the outgoing send buffer(s). During this preparation, the vocabulary of the local inverted index is traversed in order to copy the local posting list portions to the send buffer(s). Whenever a send buffer is full, the owner processor issues a blocking send operation. Blocking send operation stalls all computation on

the sender-side until the send operation is successfully completed. Upon receiving a message, each processor starts emptying its respective receive buffer by copying the received posting list portions to the final inverted index, effectively finalizing the merge of posting list portions. After the merging phase is completed, processors issue a new non-blocking receive in order to receive any remaining messages from other processors, and restart filling their send buffers.

4.5.1 1-Send (1s) versus $(K$-1)-Send (Ks) Buffer Schemes

In the 1s buffer schemes, in order to prepare messages to be sent to other processors, all posting list portions targeted to a specific processor should be put into the single send buffer prior to sending it. For a single target processor, in order to send all required posting list portions, the vocabularies of each local inverted index must be traversed once. As each processor probably requires to communicate with all other processors, preparation of the send buffers requires K-1 traversals over the local inverted index.

On the other hand, in the Ks buffer schemes, in order to prepare outgoing messages, only one traversal of the local inverted index is sufficient. In this traversal, the processor would pick any outgoing posting list portion and place it into the appropriate send buffer. Once one of the send buffers is full, the communication can commence. However, using blocking sends ultimately results in stalling the process every time a send is issued, reducing the processor utilization.

4.5.2 1-Receive (1r) versus (K-1)-Receive (Kr) Buffer Schemes

In 1r schemes, the communication memory is fairly utilized, whereas in Kr schemes, the utilization of the communication memory depends on the number of messages received by each processor and may be poor for most of the processors. In 1sKr, since there can be only K messages over the network at any time, only K of the $K \times (K-1)$ receive buffers would be actively used. In this case, $K \times (K-2)$ unused receive buffers are left idle, leaving the $(K-2) \times M$ of the total $K \times M$ memory unused. In KsKr, since there is K-1 send buffers, the processors can produce enough messages to actively use most of the $K \times (K-1)$ receive buffers, resulting with a more utilized communication memory.

In Kr schemes, since each processor has a specific receive buffer for all other processors, cycles in the communication dependency graph do not cause deadlocks. However, in 1r schemes, depending on the communication order, cycles in the communication dependency graph may cause deadlocks. To avoid these deadlocks, we can utilize non-blocking sends instead of blocking sends. Non-blocking sends allow a processor to continue processing after a send is issued without the need of waiting it to finalize, thus avoiding any possible deadlocks. However, the issued send still requires its particular send buffer to be intact. As a result, the processor should again be halted in case a local posting list is required to be written in that send buffer. For this reason, each send buffer is locked after a send, and all such buffers are probed after each messaging iteration. If a send buffer is released after a successful send, the lock is freed

allowing the processor to issue writes into that send buffer again.

In Ks1r, whenever a non-blocking send is issued, it is possible to fill other send buffers, allowing computation to overlap with communication. However, in 1s1r, deadlock avoidance via non-blocking sends may cause poor performance since there is only one send buffer and it is not possible to overwrite the contents of this buffer until the non-blocking send is completed, causing the computation to be stalled.

It is also possible to avoid deadlocks in 1s1r scheme by employing a BSP-like (148) communication/computation pattern and by ensuring that no two processors send messages to the same processor in any given communication step. In 1s1r, since K-1 traversals over the local inverted index is required for each processor, it is possible to divide the computation into K-1 traversal steps and communicate at the end of each computation step. We can also freely choose the communication order in such a scheme. By exploiting this freedom, we can find a communication schedule that avoids deadlocks. Minimizing the number of communication steps induced by this schedule corresponds to minimizing the total inversion time of the proposed BSP-like scheme.

In this work, we show that the problem of finding a communication schedule with minimum number of steps can be reduced to the "Open Shop Scheduling Problem" (OSP). In OSP, there are $|J|$ jobs and $|W|$ workstations. Each job $j_i \in J$ has to visit all workstations and perform a different task. There is an associated time $t(j_i, w_k)$ for finishing job j_i at workstation $w_k \in W$. No restrictions are placed on the execution

order of jobs and it is given that no job can be carried out simultaneously on more than one workstation.

In (54), the authors proposed an optimal algorithm to find minimum finish time in an OSP. This is achieved by constructing a bipartite graph from the jobs and workstations, iteratively finding complete matchings over this graph, and modifying the graph by decreasing edge weights of edges in the discovered matching by the smallest edge weight until no more complete matchings can be found. Finding a complete matching ensures that no two jobs are assigned to the same workstation, while no two workstations are working on the same job at any time.

The posting list exchange and merge phase of index inversion process can also be modeled using the above mentioned algorithm. In the parallel index inversion problem, each processor has to send inverted list portions to other processors. The send operation of inverted list portions corresponds to jobs in the scheduling problem. Also, each send should be received by a processor and merged into the final inverted lists. In that sense, each processor also functions as a workstation in the scheduling problem. There are two associated vertexes, one job vertex and one workstation vertex, for each processor in the bipartite graph. That is, the send responsibilities of processors constitute the jobs and the receive responsibilities of processors constitute the workstations. If a processor p_i has to send a message to processor p_j, there is an associated edge between the job vertex of p_i and workstation vertex of p_j and the number of the messages to be sent from p_i to p_j is the weight of this edge. In this model, each match found on

the constructed graph correspond to a schedule step, where finding an optimal finish time schedule defines an optimal communication schedule with least possible number of communication steps.

4.6 Experiments

4.6.1 Experimental Framework

We conducted our experiments on a realistic dataset obtained by crawling educational sites across America. The raw size of the dataset is 30 GB and contains 1,883,037 pages from 18,997 different sites. The biggest site contains 10,352 pages while average number of pages per site is 99.1. The vocabulary of the dataset consists of 3,325,075 distinct terms. There are 787,221,668 words in the dataset. The size of the inverted index generated from the dataset is 2.8 GB. For query load balancing purposes, we used a syntetically generated query log of 1,000,000 distinct queries each of which contains 1 to 7 terms. In our experiments, we used a fixed number of buckets in termBucket-to-processor assignment and set the number of buckets to 10,000.

We tested the performance of the proposed assignment schemes in two different ways: First we report the relative performances of the assignment schemes in terms of the quality metrics described in Section 4.3.2 through simulations. In simulations we theoretically compute the assignment of terms to processors and compute the storage,

query processing, and communication costs of the assignment without performing actual parallel inversion. The simulation experiments are conducted for K={4, 8, 16, 32, 64, 128} values on a Sun AMD-opteron machine with 128GB of RAM.

Second, we provide a set of experiments using actual parallel inversion runs in order to show how improvements in quality metrics relate to parallel running times. For this purpose, we developed an MPI-based parallel inversion code that can utilize each of the four communication-memory organization schemes described in Section 4.5 for a given termBucket-to-processor assignment. These second set of experiments are conducted on a 32-node PC-cluster, where each node is an Intel Pentium IV 3.0 GHz processors with 1 GB RAM connected via an interconnection network of 100 Mb/sec fast Ethernet. The total communication-memory size M is set to 5 MB in these experiments.

4.6.2 Evaluation of the Assignment Schemes

As a baseline inversion method, we implemented a random term assignment (RT) algorithm. In RT scheme, each term is assigned to a random processor without a term-to-bucket assignment. In this scheme, the global vocabulary has to be created. In order to evaluate the viability of term-to-bucket assignment and as a baseline termBucket-to-processor assignment scheme, we also implemented a random assignment (RA) algorithm which assigns buckets to processors randomly. Note that RA requires the

Table 4.1: Percent query processing load imbalance values.

K	RT	RA	MCA	BLMCA	E^1A	E^2A
4	30.5	54.3	91.4	51.6	47.8	19.6
8	55.5	86.3	115.0	78.2	74.1	24.1
16	100.4	102.8	352.1	92.4	90.1	44.8
32	319.3	233.8	457.3	167.1	123.4	61.7
40	437.2	284.9	755.8	225.6	189.3	79.8
64	602.5	503.7	1446.2	407.9	374.5	112.5
128	857.3	969.4	8456.8	821.7	682.1	216.4

least possible time to compute a termBucket-to-processor assignment while avoiding the need for global vocabulary creation, and thus it can be used to compare/analyze the merits of the proposed bucketing scheme and the assignment schemes. The performance of the proposed assignment schemes are compared against RT and RA schemes.

4.6.2.1 Simulation Results

Tables 4.1, 4.2 and 4.3 compare the performance of the assignment schemes in terms of the quality metrics described in Section 4.3.2.

Table 4.1 displays the performance of the proposed assignment schemes in optimizing the quality metric `QM1`. In the table, the query load imbalance percentages for different assignment schemes and different number of processors is presented. The query load imbalance values are calculated according to the following formula:

$$\Big(\frac{Max_{1\leq k\leq K}\{Q(p_k)\}}{(\sum_{k=1}^{K}(Q(p_k)))/K} - 1\Big) \times 100. \tag{4.11}$$

Table 4.2: Percent storage load imbalance values.

	Initial	Final					
K		RT	RA	MCA	BLMCA	E^1A	E^2A
4	13.8	4.4	12.1	38.3	0.0	2.8	5.9
8	31.9	11.7	09.9	60.0	0.1	7.2	14.1
16	38.2	18.2	27.4	66.2	1.7	9.3	23.2
32	58.4	44.1	29.6	83.0	5.4	16.1	32.5
40	66.3	32.2	37.0	77.4	6.2	17.9	33.1
64	69.0	44.7	56.6	92.2	11.5	21.7	36.0
128	81.4	65.3	94.7	95.6	15.7	32.6	45.8

Table 4.2 shows the performance of the proposed assignment schemes in optimizing quality metric `QM2`. In the table, the initial imbalances due to hash-based distribution and the final imbalances after applying the assignment schemes are presented. The storage imbalance values are computed according to the following formula:

$$\Big(\frac{Max_{1\leq k\leq K}\{S(p_k)\}}{(\sum_{k=1}^{K}(S(p_k)))/K} - 1\Big) \times 100. \tag{4.12}$$

Table 4.3 compares the communication performance of the assignment schemes in terms of average and maximum message volume to be handled by a processor during parallel index inversion. Total volume of communication required by an assignment scheme can be computed from the table by multiplying the respective average message volume value of the assignment scheme with the respective K value. Thus, the "Avg" columns of Table 4.3 indicate the performance of the assignment schemes in optimizing `QM3(a)`. The "Max" columns in Table 4.3 indicate the communication load of the maximally loaded processor and thus indicate the performance of the

Table 4.3: Message volume (send + receive) handled per processor (in terms of $\times 10^6$ postings)

	RT		RA		MCA	
K	Avg	Max	Avg	Max	Avg	Max
4	131.184	133.233	131.189	145.713	122.091	150.263
8	76.511	88.862	76.554	90.582	71.448	119.745
16	41.002	44.562	41.008	49.249	38.322	77.114
32	21.118	32.254	21.188	28.754	19.817	71.127
40	17.026	26.629	17.053	23.962	15.991	44.793
64	10.761	18.690	10.761	17.769	10.088	74.273
128	5.423	11.883	5.424	11.967	5.088	65.586

	BLMCA		E^1A		E^2A	
K	Avg	Max	Avg	Max	Avg	Max
4	127.450	128.619	129.437	134.683	131.857	131.154
8	73.402	75.974	77.562	80.327	77.385	78.229
16	39.217	43.443	43.205	44.944	42.792	42.953
32	20.283	26.025	21.218	25.788	21.047	21.695
40	16.322	20.014	17.072	20.576	17.471	18.118
64	10.339	15.421	10.981	15.222	11.201	12.354
128	5.222	10.980	5.662	10.437	7.233	8.178

assignment scheme in optimizing `QM3(b)`. The communication-load balancing performance of each assignment scheme can be evaluated by comparing the "Avg" and "Max" columns.

The comparison of RT and RA schemes relates to the effectiveness of the proposed term-to-bucket assignment. As shown in Table 4.2, RA performs slightly better than RT for $K \leq 64$. Both Tables 4.1 and 4.3 displays that RT and RA perform similarly in terms of query load balancing and communication volumes. Comparison of these two assignment schemes shows that term-to-bucket assignment prevents the global vocabulary construction without much degrading our quality metrics.

Table 4.4: Parallel inversion times (in seconds) including assignment and inverted list exchange times for different assignment and communication-memory organization schemes.

	K	RT	RA	MCA	BLMCA	E^1A	E^2A
	2	105.90	109.80	85.72	106.08	108.15	108.13
	4	71.60	69.19	81.34	68.63	69.34	68.49
1s1r	8	66.44	51.42	66.76	46.45	47.27	45.74
	16	63.00	35.89	60.82	33.04	33.65	32.48
	32	73.38	19.31	48.45	18.20	18.53	17.20
	2	105.66	109.82	86.04	105.87	107.36	107.97
	4	69.55	73.69	80.31	70.60	71.51	70.71
Ks1r	8	68.10	53.34	68.27	50.04	49.77	48.58
	16	62.59	36.66	60.30	34.32	34.70	32.91
	32	73.10	20.21	50.34	18.52	20.13	17.72
	2	105.17	109.60	86.06	106.31	108.11	108.91
	4	71.37	70.84	80.82	70.11	70.35	69.44
1sKr	8	69.60	58.13	64.97	45.78	47.63	45.22
	16	60.17	34.67	59.13	32.54	32.97	31.41
	32	73.31	20.24	48.36	18.59	19.02	18.07
	2	106.30	110.05	86.13	106.01	108.14	108.12
	4	67.25	66.79	71.89	64.50	64.08	62.51
KsKr	8	62.23	45.44	59.82	41.46	40.89	38.79
	16	57.92	31.28	54.56	29.36	28.78	26.00
	32	72.17	18.43	47.81	18.11	18.01	16.97

As seen in Table 4.1, MCA achieves significantly worse query load imbalance than all other assignment schemes. Similarly, Table 4.2 shows that MCA considerably degrades the initial storage balance. On the other hand, Table 4.3 reveals that MCA achieves the best average communication cost. These experimental findings are expected since MCA only considers the minimization of the total communication cost, disregarding storage and communication balancing.

As mentioned in Section 4.4.2, BLMCA is a modified version of MCA with added emphasis on storage balancing. As seen in Table 4.2, BLMCA achieves the best final storage balance in all instances. However, as seen in Table 4.3, the storage balance in BLMCA is achieved at the expense of increased total communication volume compared with MCA. Table 4.1 also shows that especially with increasing K, BLMCA fails in balancing query processing loads.

Table 4.1 displays that for all processor values, E^2A performs significantly better than all other assignment schemes in terms of query processing load balance. Additionally, in terms of query load imbalances, E^1A is the second best performer. As seen in Tables 4.2 and 4.3, although E^2A slightly degrades the storage balance, it performs better than the other schemes in terms of maximum communication volume handled by a processor for almost all K values (except for K=2 and 4). Although E^1A produces better storage balance than E^2A, the communication volume handled by a processor incurred by E^1A is slightly worse than BLMCA. In terms of maximum communication volume handled by a processor, E^2A achieves the best results for $K > 8$. Table 4.3

also indicates that the average and maximum communication volume values induced by E^2A are close, which shows that E^2A manages to distribute the communication load among processors evenly.

4.6.2.2 Parallel Inversion Results

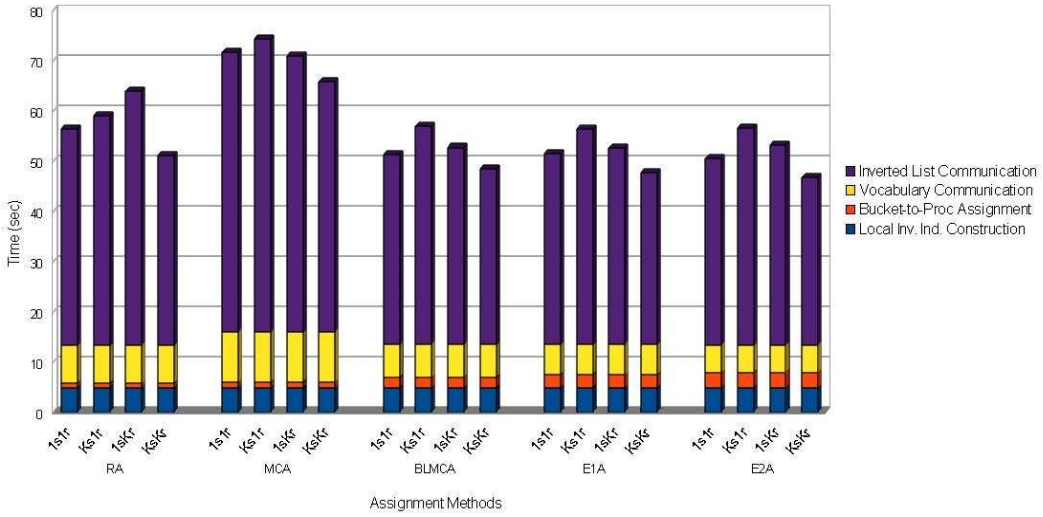


Figure 4.2: Times (secs) of various phases of the parallel inversion algorithm for different assignment and communication-memory organization schemes on $K = 8$ processors.

Table 4.4 compares the running times of our parallel inversion code for different assignment schemes. Since the creation of the local inverted indexes from local document sets is an operation prior to our inversion schemes, it is assumed that the local inverted indexes are already created. Thus, the time for converting local document collection to local inverted indexes is not included in the inversion times displayed in Table 4.4.

We provide RT scheme in order to present the benefits of using a term-to-bucket assignment. RT scheme differs from other schemes in two ways. First, in RT scheme termBucket-to-processor assignment is replaced with a term-to-processor assignment. Second, in RT scheme there is an additional phase called global vocabulary construction phase. As seen in Table 4.4, RT performs significantly worse than other assignment schemes for all K values other than $K = 2$. This indicates that our bucketing scheme has a significant impact on performance.

As seen in Table 4.4, for K=2, MCA achieves the lowest inversion time compared to the other schemes. This is because, for K=2, minimizing total communication volume also minimizes the maximum communication volume handled by a processor. However, for all K values greater than 2, MCA performs significantly worse since the maximum message volume handled by a processor for MCA is considerably higher than other assignment schemes. As seen in Table 4.4, E^2A performs considerably better than the other assignment schemes. For example for $K = 32$, E^2A performs up to 9% better than RA in terms of running time and achieves better final query load and storage balancing. The relative performance order of the assignment schemes in terms of actual inversion time values displayed in Table 4.4 are generally in concordance with the relative performance order of the assignment schemes in terms of the quality metrics displayed in Tables 4.2 and 4.3.

Figure 4.2 displays the dissection of parallel inversion time into: local inverted

index construction, termBucket-to-Processor assignment and inverted list exchange-and-merge phases for different assignment and communication-memory organization schemes on K=8 processors. For the sake of a better insight on the overall index inversion process, inverted list exchange-and-merge phase is further divided into two components. The first component is called vocabulary communication, where processors send each other the terms, in their word form, and the associated posting list sizes in an all-to-all personalized fashion. The second component is called inverted list communication, where the posting list portions are communicated between processors.

Figure 4.2 shows that for the in-memory inversion task, the construction of a global vocabulary takes considerable time. For K=8 processors, almost 35% of the total inversion time is spent on global vocabulary construction in RT scheme.

As seen in Figure 4.2, the local inverted index construction takes the same time in all schemes since local index inversion depends only on the initial data distribution. Figure 4.2 also shows that, as the complexity of the assignment schemes increases, the time required for termBucket-to-processor assignment also increases. The RA-based termBucket-to-processor assignment phase takes less than 1% of the total inversion time, whereas the E^2A-based termBucket-to-processor assignment phase takes more than 4% of the total inversion time. As the "Max" columns of Table 4.3 suggest, the time spent on vocabulary communication phase is minimum for E^2A and maximum for MCA assignment scheme.

As seen in Figure 4.2, the inverted list exchange-and-merge phase takes almost 85% of the total inversion time, thus confirming that parallel inversion is a communication-bound process. We compare and analyze the impact of different communication-memory organization schemes on this phase in the following subsection.

4.6.3 Evaluation of Communication-Memory Organization Schemes

Table 4.4 compares the running times of parallel inversion for different communication-memory organization schemes. Ks1r has the worst overall performance for all K values greater than 2. Although Ks1r avoids redundant memory reads by doing only one traversal over the local inverted lists, the use of blocking sends causes stalls and prevents overlap between communication and computation.

Although 1sKr performs better than 1s1r for $K \leq 16$, its relative performance decreases when the number of processors increases. This is due to lower memory utilization of 1sKr on higher number of processors since each processor must maintain $K-1$ receive buffers. We theorize that for higher number of processors, 1sKr would perform even worse.

For all K values greater than 2, KsKr performs superior with respect to the other communication-memory organization schemes. As the number of processors increase, the performance gap between KsKr and the other schemes increases in favor of KsKr. This is because KsKr avoids redundant traversals during the preparation of

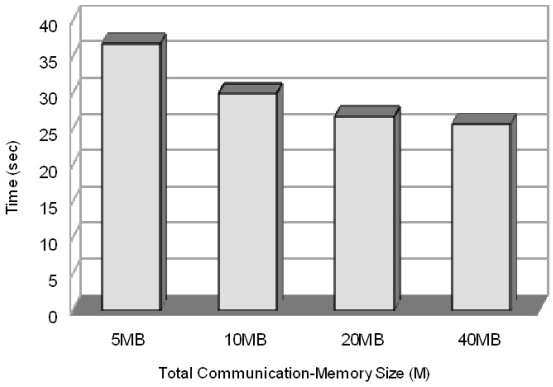


Figure 4.3: The effect of the available communication-memory size (M) on inverted list exchange-and-merge phase of a K=8 processor parallel inversion system utilizing E^2A and KsKr.

send buffers and overlaps computation with communication. For this reason, we select KsKr as the de-facto communication-memory organization scheme for the remaining experiment.

Figure 4.3 evaluates the effect of the available communication-memory size (M) on the running time of parallel inversion code utilizing the E^2A assignment scheme and KsKr communication-memory organization scheme for $K = 8$ processors. As seen in Figure 4.3, KsKr scales well with increasing communication-memory size. The ability to continue to process several send buffers without stalling allows KsKr to function relatively better with larger communication-memory sizes.

Chapter 5

Concluding Discussions

In this work, we first theorized and then analyzed two common characteristics of Web-based textual communication media. These common characteristics are: First, the web-based textual communications all contain personal attributes that can be used to exploit or identify several aspects of the communication and messages of all web-based textual communications have similar distributional properties. Second, all Web-based textual communications have similar heavy tailed distributions, for both message logs, vocabularies, and user behavior.

In order to verify our claims, rather than going over a set of previous literature work, we decided to select different web-based textual communications and examine their patterns over real-life applications. In order to do this, first, a taxonomy of the communication media with the corresponding state-of-the-art literature is provided.

Using this taxonomy, we selected three types of communication data from different classes of the web-based communications and analyzed these data.

For this purpose, we also selected three different examples of real life applications. As a first application, we selected the query collection of a real life Web search engine over a 10 day period and by the aid of our claims we proposed methods to improve caching rate over the search engine architecture. In the second application, we selected online messages on a real time chat server and examined the predictability of several attributes of both users of this chat server and their messages. As a final work, we selected a collection of Web pages and picked the inverted index creation task. We identified potential challenges on the distributed index inversion problem and using the distributional properties of the Web data we theorize a method to efficiently carry out the inversion task. Our results show that identifying and exploiting the common characteristics of common characteristics of computer mediated communications is crucial when undertaking any research challenge related to Web based communications. Our findings and conclusions can be summarized as follows:

In Chapter 2, we presented a machine learning approach to train a feature-based caching model for the query result caching problem. For training the caching models we have used in this work, we evaluated several features. These features can be grouped into five categories. These categories are: query string-based, user-based, search engine related, term frequency-based, query frequency-based, and temporal features.

Using the features that are gathered from the logs of a real life Web search engine, we trained two machine learning models. The first machine learning model trains a singleton prediction model, where each query is ranked from 0 to 1 with respect to its probability of being a singleton query or not. The second model trains a regression of each query's next arrival time, that is the estimated time of query being re-submitted to the search engine by some user. Using these two models, we constructed a caching policy, where admission and eviction decisions are given based on which queries are more likely to be observed in the near future and when.

For evaluating our models, we first examined two extreme caching organizations: a fully static cache and a fully dynamic cache. Our results show that the proposed machine learning approach improves the performance of caching in both conditions. We have also provided several tighter optimality bounds for both problems and show that the machine learning approach in fact improves the query result cache up to 11% of the possible room for improvement.

We then combine both caching organizations into one by applying the proposed machine learning approach to SDC. Our experiments indicate that the room for improvement in the caching problem is in fact smaller than what we have expected. Combining machine learned static cache and machine learned dynamic cache did not lead to the expected improvements. Although the resulting caching model still improves SDC by 7.8% of the maximum possible improvement, our results indicate a deterioration in the quality of the regression model in the dynamic cache when a static cache is

also present.

In terms of our claims, we have shown that query features, combined with a machine learning approach, can be used to improve the performance of the query result cache of a real life search engine. Moreover, our evaluations on the regression models show that, many of the most discriminative features contain non-temporal features. In that respect, our findings validate claim 1, that Web-based textual communications contain characteristic markers inherent to the text message itself or its user.

In this work, our results have also verified claim 2 by using the temporal locality within the caching problem. Caching, as the main motivation of the proposed work, is based on the fact that there is a strong temporal correlation within the query submissions to a Web search engine. In order for caching to be beneficial for a search engine, a small subset of queries should be frequent enough so that, by merely storing them in memory, the search engine can respond to most of the queries without re-computing the results. Our evaluations on the query log also indicate that almost 40% of the queries are submitted to the search engine only once, corresponding to a heavy tail distribution. Thus, we can conclude that search query logs follow a heavy tail distribution which can be categorized as a power law or log-normal distribution, which verifies that claim 2 holds for query search logs.

In Chapter 3, we examined a peer-to-peer instant messaging network. Personal and message-based features are used to predict several user- and message-based classes.

The result of this study show that personal and environmental characteristics have significant impact on ones' vocabulary use and writing style in peer-to-peer communications. In this work, it is shown that by using the word selection patterns and stylistic preferences of chat users, it is possible to predict their sociolinguistic characteristics by employing classification techniques. It is also shown that external factors such as the time of a conversation and the recipient of a message has considerable effect on the vocabulary use and writing style of an author.

The dataset used in this work also has distinguishing properties. The spontaneous nature of chatting and point-to-point nature of the chat messages makes the chat dataset quite different from any literary writing. To the best of our knowledge, in this study, for the first time in literature, the authorship analysis techniques are applied to real-time online conversations.

In terms of our claims which are presented in Chapter 1, analysis of peer-to-peer instant messaging conversations show that messages between peers contain many predictable attributes. The identity of the author of a message, the receiver of a message, the age group, connectivity domain, and gender are some examples of such attributes. Our findings clearly indicate the truth of claim 1, which is instant messaging communications contain characteristic markers inherent to their author and receiver and verifies its validity.

We believe that the outcome of this work will prove to be beneficial for many

application areas such as e-commerce and Internet security. For example, it is possible that companies supporting virtual reference services may use this method for gathering client profiles, determining a target population, and provide better and more customized service to these clients. With the growing use of Internet communication, spamming becomes a worldwide phenomenon. This application can also be used in the implementation of dynamic spam filters. Once the classifier is trained by a set of previously available spam messages, it may be possible to identify the structural properties of spam messages and detect them. The style-based approach presented in this chapter may prove to be useful for this purpose. Another direct implication is the use of our work for ensuring security within virtual groups. In most messaging services, a user is not permitted to have more than one account. Matching user profiles may prevent duplicate user accounts and can be used to detect the true source of malicious messages.

This work can be extended in several ways. First, our approach is tested using only one corpus. Application of our methods on different datasets will strengthen the findings of this work. Applying our methods to other types of electronic discourse such as emails, IRC messages, and newsgroup messages may reveal similarities between different computer-mediated communication media. Second, this work has only been tested on Turkish documents. While the applied procedure seems to be independent of the language, the effectiveness and applicability to other languages remain untested. Additionally, such a work may provide clues on common and language-independent

characteristics of electronic discourse. Third, this work relies on the "supervised learning" assumption. This means that the procedures described here are applicable only if a set of training samples is available. A framework based on unsupervised classification seems to be a natural extension of this work. In the unsupervised classification approach, the classifier generates a set of spectral classes without requiring any input. Information classes are assigned to these spectral classes afterward with user intervention. Fourth, the problem can be modeled as a probabilistic information retrieval model. Using the procedure described in this work, it may be possible to answer queries such as "find the documents that are predicted to be written during a certain period of time" or "find the documents that are possibly written by someone who has a PhD degree".

In chapter 4, a memory-based, term-partitioned parallel inverted index construction framework was examined. Several problems were identified and improvements were proposed for a parallel index inversion framework.

First, we proposed a termBucket-to-processor assignment scheme. This scheme minimizes the communication cost of local vocabularies among processors and distributes the final query processing and storage loads among all processors, allowing a finer grained parallelism. We also showed that, by using a termBucket-to-processor assignment scheme, the need to create a global vocabulary can be eliminated and all associated communications can be prevented.

Second, we developed and investigated several heuristics for generating term-to-processor assignment. The results of our experiments show that, compared to a baseline random assignment scheme, our proposed methods improved the parallel inversion times significantly while providing reasonable final query processing and storage balances.

Third, we presented and explored four different communication-memory organization schemes in order to reduce the communication time required. We also presented methods to avoid deadlocks and network congestion and commented on memory utilization of the overall system. Our results show that, splitting the communication-memory in $2 \times (K-1)$ parts yields the best results.

Fourth, Simulations and actual parallel inversion times are presented in order to give insight on our improvements. According to the observed results, we recommend the use of the E^2A scheme for termBucket-to-processor assignment, and the KsKr scheme for communication-memory organization.

Our analysis of the index inversion problem show that a naive approach for a distributed inversion problem would be too slow for any practical use. As our experiments also indicate, that is because the creation of a global vocabulary in such a system would create a serious bottleneck on processors. In order to alleviate this problem, we used our second claim; the Web page data distributions follow a heavy tail distribution, and proposed a bucketing scheme. In the proposed method the terms are hashed into a

finite number of buckets and information about these buckets are communicated between processors instead of posting list information. By applying this simple, yet effective bucketing strategy we prevent almost 35 % of the total communication and avoid global vocabulary communication all together. The success of this work also verifies that Web based communications follow similar distributions which can also be exploited in order to alleviate challenging problems on Web search engines.

This work can also be extended in several ways. First, the framework used in this work does not consider the effect of bucket processing order. For example, processing buckets in decreasing size order might present better results both in respect of final storage balance and communication costs. Second, the number of buckets is assumed to be fixed throughout this work. The scaling of our framework using different number of buckets can also be considered.

Bibliography

[1] L.A. Adamic and N. Glance. 2005. The political blogosphere and the 2004 U.S. election: divided they blog. In *LinkKDD '05 Proceedings of the 3rd International Workshop on Link Discovery*, pp. 36–43, Chicago, IL, USA.

[2] L.A. Adamic. 2008. Knowledge sharing and yahoo answers: everyone knows something. In *WWW '08 Proceedings of the 17th International Conference on World Wide Web*, pp. 665–674, Beijing, China.

[3] B. Aleman-Meza, M. Nagarajan, C. Ramakrishnan, L. Ding, P. Kolari, A.P. Sheth, I.B. Arpinar, A. Joshi, and T. Finin. 2006. Semantic analytics on social networks: experiences in addressing the problem of conflict of interest detection. In *WWW '06 Proceedings of the 15th International Conference on World Wide Web*, pp. 407–416, Edinburgh, Scotland.

[4] R. Alonso, D. Barbara, and H. Garcia-Molina. 1990. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems (TODS)*, **15**(3), 359–384.

[5] I.S. Altingovde, R. Ozcan, and O.Ulusoy. 2009. A cost-aware strategy for query result caching in search engines. In *Proceedings of the 31st European Conference on Information Retrieval Research*, pp. 628–636.

[6] I.S. Altingovde, R. Ozcan, B.B. Cambazoglu, and O. Ulusoy. 2011. A hybrid approach for dynamic result caching for search engines. In *Proceedings of the 33rd European Conference on Information Retrieval research*, pp. 510–516.

[7] S. Argamon, M. Saric, and S.S. Stein. 2003. Style mining of electronic messages for multiple authorship discrimination: first results. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 475–480, Washington, DC, USA.

[8] D. Avrahami and S.E. Hudson. 2006. Communication characteristics of instant messaging: effects and predictions of interpersonal relationships. In *CSCW '06 Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work* , pp 505–514, Banff, Alberta, Canada.

[9] C. Aykanat, B.B. Cambazoglu, F. Findik, and T. Kurc. 2007. Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids. *Journal of Parallel Distributed Computing*, **67**,77–99.

[10] R.H. Baayen, H. van Halteren, and F.J. Tweedie. 1996. Outside the cave of shadows: using syntactic annotation to enhanceauthorship attribution. *Literary and Linguistic Computing*, **11**(3), 121–132.

[11] E. Backer and P. van Kranenburg. 2004. Musical style recognition - a quantitative approach. In *Proceedings of the Conference on Interdisciplinary Musicology (CIM04)*, Graz, Austria.

[12] R. Baeza-Yates and F. Saint-Jean. 2003. A three level search engine index based in query log distribution. In *SPIRE'03 Proceedings of the 10th String Processing and Information Retrieval*, pp. 56–65, Manaus, Brazil.

[13] R. Baeza-Yates, F. Junquiera, V. Plachouras, and H.F. Witschel. 2007. Admission policies for caches of search engine results. In *SPIRE'07 Proceedings of the 14th International Conference on String Processing and Information Retrieval*, pp. 74–85, Berlin, Heidelberg, Germany.

[14] R. Baeza-Yates, A. Gionis, F. Junquiera, V. Murdock, V. Plachouras, and F. Silvestri. 2007. The impact of caching on search engines. In *SIGIR '07 Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 183–190, Amsterdam, The Netherlands.

[15] R. Baeza-Yates, A. Ginois, F.P. Junquiera, V. Murdock, V. Plachouras, and F. Silvestri. 2008. Design trade-offs for search engine caching. *ACM Transactions on the Web (TWEB)*, **2**(4), 20–28.

[16] L. Backstorm, D. Hottenlocher, J. Kleinberg, and X. Lan 2006. Group formation

in large social networks: membership, growth, and evolution. In *KDD '06 Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pp. 44–54, Philadelphia, PA, USA.

[17] N. Bansal and N. Koudas 2007. BlogScope: spatio-temporal analysis of the blogosphere. In *WWW '07 Proceedings of the 16th International Conference on World Wide Web*, pp. 1269–1270, Banff, Alberta, Canada.

[18] L. Barroso, J. Dean, and U. Holzle. 2003. Web search for a planet: The google cluster architecture. *Micro, IEEE*, **23**(2), 22–28.

[19] J. Baumes, M. Goldberg, M. Hayvanovych, M. Magdon-Ismail, W. Wallace, and M. Zaki 2006. Finding hidden group structure in a stream of communications. In *Intelligence and Security Informatics*, pp. 201–212.

[20] F. Benevenueto, T. Rodrigues, V. Almeida, J. Almeida, and M. Goncalves 2009. Detecting spammers and content promoters in online video social networks. In *SIGIR '09 Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 620–627, Boston, MA, USA.

[21] J.N.G. Binongo and M.W.A. Smith. 1999. The application of principal component analysis to stylometry. *Literary and Linguistic Computing*, **11**(3), 121–131.

[22] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. 2006. Mining

email social networks In *MSR '06 Proceedings of the 2006 International Workshop on Mining Software Repositories*, pp. 137–143, Shangai, China.

[23] R.H. Bisseling and W. Meesen. 2005. Communication balancing in parallel sparse matrix-vector multiplication. *Electronic Transactions on Numerical Analysis*. Special Issue on Combinatorial Scientific Computing, **21**, 47–65.

[24] R. Blanco, E. Bortinkov, F. Junqueira, R. Lempel, L. Telloli, and H. Zaragoza. 2010. Caching search engine results over incremental indices. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 82–89.

[25] E. Bortinkov, R. Lempel, and K. Vornovitski. 2011. Caching for real time search. In *ECIR'11*, pp. 104–116.

[26] S. Brin, and L. Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th International Conference on World Wide Web*, Brisbane, 14–18 April, pp. 107–117, Australia.

[27] E. Brynjolfsson, H.J. Yu, and D.S. Michael. 2003. Consumer surplus in the digital economy: estimating the value of increased product variety at online booksellers. *Management Science*, **49**11.

[28] J. Burrows. 1987. *Computation into criticism: A study of Jane Austen's novels and an experiment in method*. Oxford: Clarendon Press.

[29] S. Buttcher, and C.L.A. Clarke. 2006. A hybrid approach to index maintenance in dynamic text retrieval systems. In*Advances in Information Retrieval 28th European Conference on IR Research*, pp. 229–240, 10–12 April, London, UK.

[30] B.B. Cambazoglu and C. Aykanat. 2005. Harbinger Machine Learning Toolkit manual. Technical Report BU-CE-0503, Bilkent University, Computer Engineering Department, Ankara.

[31] B.B. Cambazoglu, F.P. Junquiera, and V. Plachouras. 2010. A refreshing perspective of search engine caching. In *WWW '10 Proceedings of the 19th International Conference on World Wide Web*, pp. 181–190, Raleigh, North Carolina, USA.

[32] B.B. Cambazoglu, I.S. Altingovde, R. Ozcan, and O.Ulusoy. 2012. Cache-based query processing for search engines. *ACM Transactions on the Web (TWEB)*, **6**(4), no. 14.

[33] F. Can and J.M. Patton. 2004. Change of writing style with time. *Computers and Humanities*, **38**(1), 61–82.

[34] A. Cevahir, C. Aykanat, A. Turk, A., and B.B. Cambazoglu. 2010. Site-based partitioning and repartitioning techniques for parallel pagerank computation. *IEEE Transactions on Parallel and Distributed Systems*, **22**(5), 786–802.

[35] J. Cho and H. Garcia-Molina. 2000. The evolution of the web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on VLDB*, Cairo, 10–14 September, pp. 200-209, Egypt.

[36] J. Cho and H. Garcia-Molina. 2002. Parallel crawlers. In *Procedings of the 11th International Conference on World Wide Web*, Honolulu, Hawaii, 7–11 May, pp. 124–135, USA.

[37] L. Chen, L. Zhang, F. Jing, K.F. Deng, and W.Y. Ma. 2006. Ranking web objects from multiple communities. In *CIKM '06 Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pp. 377-386, Arlington, Virginia, USA.

[38] E.G. Coffman and P.J.Denning. 1973. Operating Systems Theory. Prentice-Hall, New Jersey, Ch. 6, pp. 241–283.

[39] M.W. Corney, A. Anderson, G. Mohay, and D.O. Vel. 2001. Mining email content for author identification forensics. *SIGMOD Record Web Edition*, **30**(4), 55–64.

[40] M.W. Corney. 2003. Analyzing E-mail text authorship for forensic purposes. *M.S. Thesis*. Queensland University of Technology.

[41] T. Curk, J. Demsar, Q. Xu, G. Leban, U. Petrovic, I. Bratko, G. Shaulsky, and B. Zupan. 2005. Microarray data mining with visual programming *Bioinformatics*, **21**(3), 396–8.

[42] M.H. Dickey, G. Burnett, K.M. Chudoba, and M.M. Kazmer. 2007. Do you read me? Perspective making and perspective taking in chat communities. *Journal of the Association for Information Systems*, **8**(1), 47–70.

[43] C. Dwyer, S.R. Hiltz, and K. Passerini 2007. Trust and privacy concern within social networking sites: a comparison of Facebook and MySpace,. In *Proceedings of AMCIS 2007*.

[44] W.E.Y. Elliot and R.J. Valenza. 1991. Was the Earl of Oxford the true Shakespeare? A computer aided analysis. *Notes and Queries*, **236**, 501–506.

[45] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. 2006. Boosting the performance of web search engines: caching and prefetching query results by exploiting historical usage data. *ACM Transactions on Information Systems (TOIS)*, **24**(1), 51–78.

[46] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, **9**, 1871–1874.

[47] D.W. Foster. 2000. *Author unknown: on the trail of anonymous*. New York: Henry Holt.

[48] O. Freider and H.T. Siegelmann. 1991. On the allocation of documents in multiprocessor information systems. In *Proceedings of the 14th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Illinois, 13–16 October, pp 230–239. USA.

[49] J.H. Friedman. 2002. Sthocastic gradient boosting. *Computational Statistics and Data Analysis*, **38**(4), 367–378.

[50] G. Gan and T. Suel. 2009. Improved techniques for result caching in web search engines. In *WWW '09 Proceedings of the 18th International Conference on World Wide Web*, New York, pp. 431–440, USA.

[51] G. Geisler and S. Burns. 2007. Tagging video: conventions and strategies of the YouTube community. In *JCDL '07 Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries*, pp. 480, Vancouver, BC, Canada.

[52] V. Gomez, A. Kaltenbrunner, and V. Lopez 2008. Statistical analysis of the social network and discussion threads in slashdot. In *WWW '08 Proceedings of the 17th International Conference on World Wide Web*, pp. 645–654, Beijing, China.

[53] http://www.google.com, Retrieved in January 2012.

[54] T. Gonzales and S. Sahni. 1976. Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, **23**(4), 665–679.

[55] N. Graham, G. Hirst, and B. Marthi. 2005. Segmenting documents by stylistic character. *Natural Language Engineering*, **11**(4), 397–415.

[56] K. Hakuta. 1991. Bilingualism as a gift. In *Stanford Center for Chicano Research Working Paper Series* **33**.

[57] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten 2009. he WEKA Data Mining Software: An Update. In *SIGKDD Explorations*, Volume 11, Issue 1.

[58] E. Han, G. Karypis, and V. Kumar. 2001. Text categorization using weight adjusted k-nearest neighbor classification. In *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 53–65.

[59] S. Heinz and J. Zobel. 2003. Efficient single-pass index construction for text databases. *Journal of American Society for Information Science and Technology*, **54**(8), 713–729.

[60] S.C. Herring and J.C. Paolillo. 2006. Gender and genre variations in Weblogs *Journal of Sociolinguistics*, **10**(4), 439–459.

[61] S.C. Herring and B. Danet (Eds.) 2007. *Multilingual Internet: Language, Culture, and Communication Online*, New York: Oxford University Press.

[62] D.I. Holmes. 1985. Analysis of literary style - a review. *Journal of the Royal Statistical Society*, **148**(4), 328–341.

[63] D.I. Holmes. 1994. Authorship attribution. *Computers and the Humanities*, **28**(2), 87–106.

[64] D.I. Holmes and R. Forsyth. 1995. The Federalist revisited: new directions in authorship attribution. *Literary and Linguistic Computing*, **10**(2), 111–127.

[65] S. Hota, S. Argamon, and R.Chung. 2006. Gender in Shakespeare: automatic stylistics gender classification using syntactic, lexical, and lemma features. In *Chicago Colloquium on Digital Humanities and Computer Science*, Chicago, Illinois.

[66] J. Huang, K.M. Thornton, and E.M. Efhimiadis. 2010. Conversational tagging in twitter. In *HT '10 Proceedings of the 21st ACM Conference on Hypertext and Hypermedia*, pp. 173–178, Toronto, Canada.

[67] A. Java, X. Song, T. Finin, and B. Tseng 2007. Why we twitter: understanding microblogging usage and communities. In *WebKDD/SNA-KDD '07 Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, pp. 56–65, San Jose, California, USA.

[68] D. Jensen and J. Neville 2002. Data mining in social networks. In *ational Academy of Sciences Symposium on Dynamic Social Network Modeling and Analysis*.

[69] B.S. Jeong and E. Omiecinski. 1995. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, **6**(2), 142–153.

[70] T. Joachims. 1998. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of 10th European Conference on Machine Learning (ECML-98)*, pp. 137–142, Heidelberg, Germany.

[71] E. Jonsson. 1998. Electronic discourse - on speech and writing on the Internet. Retrieved June 24, 2006, from www.ludd.luth/ jonsson/D-essay/index.html.

[72] P. Juola and R.H. Baayen. 2005. A controlled-corpus experiment in authorship identification by cross-entropy. *Literary and Linguistic Computing*, **20**(1), 59–67.

[73] J. Karlgren and D. Cutting. 1994. Recognizing text genres with simple metrics using discriminant analysis. In *Proceedings of the 15th International Conference on Computational Linguistics - Volume 2*, pp. 1071–1075, Kyoto, Japan.

[74] B. Kessler, G. Nunberg, and H. Schutze. 1997. Automatic detection of text genre. In *Proceedings of the 35th Annual Meeting on Association for Computational Linguistics*, pp. 32–38, Madrid, Spain.

[75] D.V. Khmelev and F.J. Tweedie. 2001. Using Markov chains for identification of writers. *Literary and Linguistic Computing*, **16**(3), 299–307.

[76] B. Kjell. 1994. Authorship attribution of text samples using neural networks and Bayesian classifiers. In *IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, Texas.

[77] M. Koppel, S. Argamon, and A.R. Shimoni. 2002. Automatically categorizing written texts by author gender. *Literary and Linguistic Computing*, **17**(4), 401–412.

[78] G. Kossinets, J. Kleinberg, and D. Watts. 2008. The structure of information pathways in a social communication network. In *KDD '08 Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 435–443, Las Vegas, Nevada, USA.

[79] I. Krusl and E.H. Spafford. 1997. Authorship analysis: identifying the author of a program. *Computers and Security*, **16**(3), 233–257.

[80] M. Kubat and S. Matwin. 1997. Addressing the curse of imbalanced data sets: one-sided sampling. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 179–186, Nashville, Tennessee.

[81] T. Kucukyilmaz, B.B. Cambazoglu, C. Aykanat, and F. Can. 2006. Chat mining for gender prediction. In *Proceedings of the Fourth Biennial Conference on Advances in Information Sciences*, pp. 274–284, Izmir, Turkey.

[82] T. Kucukyilmaz, B.B. Cambazoglu, C. Aykanat, and F. Can. 2008. Chat mining: Predicting user and message attributes in computer-mediated communication. *Information Processing and Management*, **44**(4), 1448–1466.

[83] T. Kucukyilmaz, A. Turk, and C. Aykanat. 2011. Memory resident parallel inverted index construction. In *Proceedings of the 26th International Symposium on Computer and Information Sciences*, London, 26–28 September, UK.

[84] R. Kumar 1998. Recommendation systems: a probabilistic analysis. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pp. 664–673, San Jose, California, USA.

[85] J. Kungenis, A. Lommatzsch, and C. Bauckhage 2009. The slashdot zoo: mining a social network with negative edges. In *WWW '09 Proceedings of the 18th International Conference on World Wide Web*, pp. 741–750, Madrid, Spain.

[86] W. Lam, M.E. Ruiz, and P. Srinivasan. 1999. Automatic text categorization

and its applications to text retrieval. *IEEE Transactions on Knowledge and Data Engineering*, **11**(6), 865–879.

[87] R. Lempel and S. Moran 2003. Predictive caching and prefetching of query results in search engines. In *WWW '03 Proceedings of the 12th International Conference on World Wide Web*, pp. 19–28, Budapest, Hungary.

[88] R. Lempel and S. Moran. 2004. Optimizing result prefetching in web search engines with segmented indices. *ACM Transactions on Internet Technology*, **4**(1), 31–59.

[89] J. Leskovec and E. Horvitz 2008. Planetary-scale views on a large instant-messaging network. In *Proceedings of the 17th International Conference on World Wide Web*, Beijing, China.

[90] J. Leskovec, L. Backstorm, R. Kumar, and A. Tomkins 2008. Microscopic evolution of social networks. In *KDD '08 Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.462–470, Las Vegas, Nevada, USA.

[91] Lester, N., Zobel, J., and Williams, H. 2006. Efficient online index maintenance for contigious inverted lists. *Information Processing and Management*, **42**(4), 916–933.

[92] S. Levitan and S. Argamon. 2006. Fixing the Federalist: correcting results and

evaluating editions for automated attribution. In *Digital Humanities*, pp. 323–328, Paris, France.

[93] K. Lewis, J. Kaufman, M. Gonzalez, A. Wimmer, and N. Cristakis 2008. Tastes, ties, and time: A new social network dataset using Facebook.com. *Social Networks*, **30**(4), 330–342.

[94] 2012 Obtained from *http://www.livinginternet.com/i/iw_dns_name.htm* 25.11.2012.

[95] X. Long and T. Suel. 2006. Three-Level Caching for Efficient Query Processing in Large Web Search Engines. *World Wide Web*, **9**(4), 369–395.

[96] H. Love. 2002. *Attributing authorship: an introduction*. Cambridge: Cambridge University Press.

[97] Z. Lu and K.S. McKinley. 2000. Partial collection replication versus caching for information retrieval systems. In *SIGIR '00 Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 248–255, Athens, Greece.

[98] E.P. Markatos. 2001. On caching search engine query results. *Computer Communications*, **24**(2), 137–143.

[99] M. Marin, V. Gil-Costa, and C. Gomez-Pantoja. 2010. New caching techniques for web search engines. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 215–226, New York, NY, USA.

[100] T.P. Martin, I.A. MacLeod, J.I. Russel, K. Leese, and B. Foster. 1990. A case study of caching strategies for a distributed full text retrieval systems. *Information Processing and Management*, **26**(2), 227–247.

[101] T.P. Martin and J.I. Russel. 1991. Data caching strategies for distributed full text retrieval systems. *Information Systems*, **16**(1), 1–11.

[102] Y. Matsuo and H. Yamamoto 2009. Community gravity: measuring bidirectional effects by trust and rating on online social networks. In *WWW '09 Proceedings of the 18th International Conference on World Wide Web*, pp. 751–760, Madrid, Spain.

[103] A. McCallum and K. Nigam. 1998. A comparison of event models for naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, Madison, Wisconsin.

[104] S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina. 2001. Building a distributed full-text index for the web. *ACM Transactions on Information Systems*, **19**, 217–241.

[105] T. Merriam and R. Matthews. 1994. Neural computation in stylometry II: an application to the works of Shakespeare and Marlowe. *Literary and Linguistic Computing*, **9**, 1–6.

[106] A. Moffat and T.A.H. Bell. 1995. In situ generation of compressed inverted files. *Journal of the American Society for Information Science*, **45**, 537–550.

[107] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. 2005. A pipelined architecture for distributed text query evaluation. *Information Retrieval*, **10**(3), 205–231.

[108] A. Moffat, W. Webber, and J. Zobel. 2006. Load balancing for term-distributed parallel retrieval. In *Proceedings of the 29th International ACM SIGIR Conference on Research and Development in IR*, Seattle, Washington, 6–11 August, pp. 348–355. USA.

[109] F. Mosteller and D.L. Wallace. 1964. *Inference and disputed authorship: the Federalist*. Reading: Addison-Wesley.

[110] R. Ozcan, I.S. Altingovde, O. Ulusoy. 2004. Static query result caching revisited. In *WWW '08 Proceedings of the 17th International Conference on World Wide Web*, pp. 1169–1170, Beijing, China.

[111] R. Ozcan, I.S. Altingovde, B.B. Cambazoglu, F.P. Junqueira, and O. Ulusoy 2012. A five-level static cache architecture for web search engines. *Information Processing and Management*, **48**(5), 828–840.

[112] S. Pandit, D.H. Chau, S. Wang, and C. Faloutsos. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW '07 Proceedings of the 16th International Conference on World Wide Web*, pp. 201–210, Bannf, Alberta, Canada.

[113] G. Pass, A. Chowdhury, and C. Torgeson. 2006. A picture of search. In *International Conference on Scalable Information Systems*.

[114] J.M. Patton and F. Can. 2004. A stylometric analysis of Yasar Kemal's Ince Memed tetralogy. *Computers and the Humanities*, **38**(4), 457–467.

[115] S. Petrovic, M. Osborne, and V. Lavrenko 2010. Streaming first story detection with application to Twitter. In *HLT '10 Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 181–189, Los Angeles, California, USA.

[116] S. Podliping and L. Bozsormenyi. 2003. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, **35**(4), 374–398.

[117] M. Pollatschek and Y.T. Radday. 1981. Vocabulary richness and concentration in Hebrew biblical literature. *Association for Literary and Linguistic Computing Bulletin*, **8**, 217–231.

[118] M.L. Radford. 2005. Encountering virtual users: A qualitative investigation of interpersonal communication in chat reference. *Journal of the American Society for Information Science and Technology*, **57**(8), 1046–1059.

[119] M.L. Radford and L.S. Connaway. 2007. "Screenagers" and live chat reference: Living up to the promise. *Scan*, **26**(1), 31–39.

[120] B.A. Ribeiro-Neto, J.P. Kitajima, G. Navarro, C.R.G. Sant'Ana, and N. Ziviani.

1998. Parallel generation of inverted files for distributed text collections. In *Proceedings of the 18th International Conference of the Chilean Computer Science Society*, Antofagasta, 12–14 November, pp. 149. Chile.

[121] B.A. Ribeiro-Neto, E.S. Moura, M.S. Neubert, and N. Ziviani. 1999. Efficient distributed algorithms to build inverted files. In *Proceedings of the 22nd International ACM SIGIR Conference on Research and Development in IR*, Berkeley, CA, 15–19 August, pp. 105–112. USA.

[122] J. Rudman. 1998. The state of authorship attribution studies: some problems and solutions. *Computers and the Humanities*, **31**(4), 351–365.

[123] J. Rzeszortaski and A. Kittur. 2012. Learning from history: predicting reverted work at the word level in Wikipedia. In *CSCW '12 Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pp. 437–440, Seattle, Washington, USA.

[124] M. Sabordo, S.Y. Chai, M.J. Berryman, and D. Abbott. 2005. Who wrote the "Letter to the Hebrews"?: data mining for detection of text authorship. In *Proceedings of the SPIE*, **5649**, 513-524.

[125] S. Alici, I.S. Altingovde, R. Ozcan, B.B. Cambazoglu, and O. Ulusoy. 2011. Timestamp-based result cache invalidation for Web search engines. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 973–982.

[126] S. Alici, I.S. Altingovde, R. Ozcan, B.B. Cambazoglu, and O. Ulusoy. 2012 Adaptive time-to-live strategies for query result caching in Web search engines. In *ECIR'12*, pp. 401–412.

[127] G. Salton and M.J. McGill. 1983. *Introduction to Modern Information Retrieval*: McGraw-Hill.

[128] G. Salton. 1989. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing, Boston, MA, USA.

[129] P.C. Saravia, E.S. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto. 2001. Rank-preserving two-level caching for scalable search engines. In *SIGIR '01 Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 51–58, New Orleans, Louisiana, USA.

[130] M.F. Schwartz and D.C.M. Wood. 1993. Discovering shared interests using graph analysis. *Communications of the ACM*, **36**(8), 78–89, 1993.

[131] F. Sebastiani. 2002. Machine learning in automated text categorization. *ACM Computing Surveys*, **34**(1), 1–47.

[132] S. Sen, J. Vig, and J. Reidl 2009. Tagommenders: connecting users to items through tags. In *WWW '09 Proceedings of the 18th International Conference on World Wide Web*, pp. 671–680, Madrid, Spain.

[133] P. Simpson and R. Alonso. 1987. Data cashing in IR systems. In *SIGIR '87 Proceedings of the 10th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 296–305, New Orleans, Louisiana, USA.

[134] G. Skobeltsyn, F. Junqueira, V. Plachouras, and R. Baeza-Yates. 2008. ResIn: a combination of results caching and index pruning for high-performance web search engines. In *SIGIR '08 Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 131–138, Singapore, Singapore.

[135] O. Sornil. 2001. *Parallel inverted index for large scale dynamic digital libraries*. Ph.D. Thesis, Virginial Polytechnic Institute and State University.

[136] E.H. Spafford and E.H. Weeber. 1993. Software forensics: can we track code to its authors? *Computers and Security*, **12**(6), 585–595.

[137] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, M. Demirbas. 2010. Short text classification in twitter to improve information filtering. In *SIGIR '10 Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 841–842, Geneva, Switzerland.

[138] E. Stamatos, N. Fakotakis, and G. Kokkotakis. 2000. Automatic text categorization in terms of genre and author. *Computational Linguistics*, **26**(4), 471–495.

[139] M. Thelwall. 2008. Social networks, gender, and friending: An analysis of

MySpace member profiles. *Journal of the American Society for Information Science and Technology*, **59**(8), 1321–1330.

[140] M. Thelwall, D. Wilkinson, and S. Uppal. 2010. Data mining emotion in social network communication: Gender differences in MySpace. *Journal of the American Society for Information Science and Technology*, **61**(1), 190–199.

[141] R. Thomson and T. Murachver. 2001. Predicting gender from electronic discourse. *British Journal of Social Psychology*, **40**(2), 193–208.

[142] A. Tomasic,H. Garcia-Mollina. 1993. Caching and database scaling in distributed shared-nothing information retrieval systems *ACM Sigmod Record*, **22**(2), 129–138.

[143] Y. Tsegay, A. Turpin, and J. Zobel. 2007. Dynamic index pruning for effective caching. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*. pp. 987–990, New York, USA.

[144] Y. Tsuboi and Y. Matsumoto. 2002. Authorship identification for heterogeneous documents. *M.S. Thesis*. Nara Institute of Science and Technology.

[145] M.T. Turell. 2004. Textual kidnapping revisited: the case of plagiarism in literary translation. *The International Journal of Speech, Language and the Law*, **11**(1), 1–26.

[146] F.J. Tweedie, S. Singh, and D.I. Holmes. 1996. Neural network applications in stylometry: the federalist papers. *Computers and the Humanities*, **30**(1), 1–10.

[147] B. Uçar and C. Aykanat. 2004. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. *SIAM Journal of Scientific Computing*, **25**(6),1837–1859.

[148] L.G. Valiant. 1990. A bridging model for parallel computation. *Communications of the ACM*, **33**(8), 103–111.

[149] D.O. Vel, M. Corney, A. Anderson, and G. Mohay. 2002. Language and gender author cohort analysis of e-mail for computer forensics. In *Second Digital Forensics Research Workshop*, Syracuse, USA.

[150] B. Viswanath, A. Mislove, M. Cha, and K.P. Gummadi 2009. On the evolution of user interaction in Facebook. In *WOSN '09 Proceedings of the 2nd ACM Workshop on Online Social Networks*, pp. 37–42, Barcelona, Spain.

[151] J.B. Walther, U. Bunz, and N.N. Bazarova. 2005. Rules of virtual groups. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii.

[152] Wikipedia. 2007. Emoticon.

Retrieved October 23, 2007, from http://en.wikipedia.org/wiki/Emoticon.

[153] F. Wilcoxson. 1945. Individual comparisons by ranking methods. *Biometrics*, **1**, 80–83.

[154] I.H. Witten, A. Moffat, and T.C. Bell. 1999. *Managing Gigabytes : Compressing and Indexing Documents and Images*. Von Nostrand Reinhold, San Francisco, CA, 2. edition.

[155] Z. Xiao, L. Guo, and J. Tracey. 2007. Understanding instant messaging traffic characteristics. In *7th International Conference on Distributed Computing Systems*, Toronto, Canada, June 25–27, 2007.

[156] Y. Xie and D. O'Hallaron. 2002. Locality in search engine queries and its implications for caching. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 1238–1247.

[157] http://www.yahoo.com

[158] Y. Yang and J.O. Pedersen. 1997. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 412–420, Nashville, Tennessee, USA.

[159] J.Ye, J.H. Chow, J. Chen, and Z. Zheng 2009. Stochastic gradient boosted distributed decision trees. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pp. 2061–2064, New York, N.Y., USA.

[160] A. Zelenkauskatie and S.C. Herring. 2006. Gender encoding of typographical elements in Lithuanian and Croatian IRC In *Proceedings of Cultural Attitudes Towards Technology and Communication*, pp. 474–489, Tartu, Estonia.

[161] T. Zesch, C. Muller, and I. Gurevych. 2008. Extracting lexical semantic knowledge from Wikipedia and Wiktionary. In *LREC'08 Proceedings of the Sixth International Conference on Language Resources and Evaluation*, Marrakech, Morocco.

[162] J. Zhang, X. Long, and T. Suel. 2008. Performance of compressed inverted list caching in search engines. In *WWW '08 Proceedings of the 17th International Conference on World Wide Web*, pp. 387–396. New York, USA.

[163] R. Zheng, J. Li, H. Chen, and Z. Huang. 2006. A framework for authorship identification of online messages: writing-style and classification techniques. *Journal of the American Society for Information Science and Technology*, **57**(3), 378–393.

[164] G. Zipf. 1932. Selective Studies and the Principle of Relative Frequency in Language. Harvard University Press, MA.

[165] J. Zobel, A. Moffat, and K. Ramamohanarao. 1998. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, **23**, 453–490.